Packt

异步图书
www.epubit.com.cn

用Python探索Web机器学习系统开发　让系统做出更加聪明的预测

机器学习Web应用

Machine Learning for the Web

eBay公司EU Analytics部门负责人Davide Cervellin作序

[意] Andrea Isoni 著

杜春晓 译


中国工信出版集团


人民邮电出版社
POSTS & TELECOM PRESS



机器学习 Web 应用

Machine Learning for the Web

[意] Andrea Isoni 著
杜春晓 译

人 民 邮 电 出 版 社
北 京

图书在版编目（ＣＩＰ）数据

机器学习Web应用 / （意）爱索尼克（Andrea Isoni）著；杜春晓译. -- 北京：人民邮电出版社，2017.8
ISBN 978-7-115-45852-0

Ⅰ. ①机… Ⅱ. ①爱… ②杜… Ⅲ. ①机器学习 Ⅳ. ①TP181

中国版本图书馆CIP数据核字(2017)第141915号

版权声明

!

◆! 著　　　　[意] Andrea Isoni
译　　　　杜春晓
责任编辑　陈冀康
责任印制　焦志炜

◆! 人民邮电出版社出版发行!!!! 北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn!
网址　http://www.ptpress.com.cn!
北京鑫正大印刷有限公司印刷

◆! 开本：800×1000　1/16
印张：14.5
字数：280 千字　　　　2017 年 8 月第 1 版
印数：1 – 2 400 册　　　2017 年 8 月北京第 1 次印刷

著作权合同登记号　图字：01-2016-8595 号!

定价：59.00 元

读者服务热线：(010) 81055410　印装质量热线：(010) 81055316!
反盗版热线：(010) 81055315
广告经营许可证：京东工商广登字 20170147 号

内容提要

机器学习可用来处理由用户产生的、数量不断增长的 Web 数据。

本书讲解如何用 Python 语言、Django 框架开发一款 Web 商业应用，以及如何用一些现成的库和工具（sklearn、scipy、nltk 和 Django 等）处理和分析应用所生成或使用的数据。本书不仅涉及机器学习的核心概念，还介绍了如何将数据部署到用 Django 框架开发的 Web 应用，包括 Web、文档和服务器端数据的挖掘和推荐引擎的搭建方法。

本书适合有志于成为或刚刚成为数据科学家的读者学习，也适合对机器学习、Web 数据挖掘等技术实践感兴趣的读者参考阅读。

序

机器学习是什么？2016 年，无论参加大会、研讨会，还是接受采访，很多人都让我给机器学习下个定义。人们对机器学习是什么，抱有诸多疑问。理解这一新鲜事物可能为生活带来的潜在影响以及它日后对我们有何种意义之前，天性要求我们先给出其定义。

跟其他陡升为显学的学科类似，机器学习并不是新生事物。科学社区多年来一直致力于研制算法，实现重复性工作的自动化。参数固定的算法叫作静态算法，其输出是可预测的，输出只是输入变量的函数。还有一种情况，算法的参数是动态变化的，算法的输出是外部因素（最常见的是同一算法先前的输出）的函数，这种算法叫作动态算法，其输出不仅仅是输入变量的函数。动态算法是机器学习的支柱：从先前迭代生成的数据中，学习到一组规则，以改善之后的输出。

科学家、开发人员和工程师研究和使用模糊逻辑、神经网络和其他类型的机器学习技术已有多个年头，但直到今天，随着机器学习应用离开实验室，进入市场营销、销售和金融行业，这门学科才流行起来，基本上来讲，需要重复执行相同运算的活动都可以受益于机器学习。

机器学习的影响很容易理解，它将给我们的社会带来巨大冲击。关于下一个 5 到 10 年，机器学习将给我们带来什么，我能想到的最佳描述方式是：不妨回想工业革命时期发生了什么。蒸汽机发明之前，很多人从事高度重复性的体力工作。为了赚取少得不能再少的工资，他们往往要冒着生命危险或以牺牲健康为代价。工业革命出现后，社会得以发展，机器接管了生产过程的重要步骤，这带来了产量的增加，并且产出的可预测性更强和更稳定。与之相应的是，产品质量的提升和新工种的出现，操控机器这类新兴的工作取代了体力劳动。我们将造物的责任委托给由我们设计和发明的工具，这在人类历史上可是第一次。

机器学习将以相同的方式，改变执行数据运算的方式，减少人工干预的需要，将优化的工作交给机器和算法。数据处理人员将不再直接控制数据，而是通过控制算法间接控制数据。因此，运算的执行速度将会变得更快，更少的人将能控制规模更大的数据集，错误将会减少，从而结果的稳定性更高和可预测性更强。跟其他对我们生活产生重大影响的事物一样，爱慕和憎恶它的人都有。爱慕者称赞机器学习为他们生活带来便利；憎恶者批评，机器学习方法要有效，需要大量的迭代，因此需要大量数据。而通常来讲，我们“喂给”算法的数据可是我们的个人信息[①]。

事实上，机器学习作为一种工具得以迅速发展，其主要应用在于提升市场营销和顾客支持的效率。为顾客提供个性化服务，促使他们购买而不只是浏览，或让他们高兴而不是失望，需要对顾客有着深入的理解。

例如，就市场营销而言，如今市场营销人员开始考虑位置、设备、购买历史、访问过的网站、天气状况等信息（仅举几个例子）来决定公司是否向一组特定顾客展示广告。

通过电视或报纸这样无法追踪的媒体传播营销信息的日子已然成为遥远的过去。如今，市场营销人员希望知道谁点击和购买了他们商品等一切信息，他们好优化创意和投入，合理分配预算，以充分利用他们手中的资源。这就要求提供前所未有的高度个性化服务，若使用合理，可以让顾客感到他们是受尊重的个体而不只是某一社会人口学分组的一部分。

机器学习既吸引人又充满挑战，但无疑下一个十年的赢家，将会是那些能够理解非结构化数据，并且能够基于这些数据以可扩展的方式做出决策的公司或个人：除了机器学习，我还没有看到哪种方式能实现这样的伟业。

Andrea Isoni 的这本书朝这个世界迈出了一步；读它就好像是向下窥视兔子的洞穴，你能从中看到用机器学习技术实现的几个应用，作者将机器学习技术整合到 Web 应用中。访问用机器学习技术创建的个性化服务网站，顾客能从中体验到为他们个人提供的优化过的服务。

如果你想为日后的职业生涯提前做好准备，该书是你必须要读的；下一个十年跟数据打交道的任何人若想成功的话，都需要熟练掌握这些技术。

Davide Cervellin，@ingdave

eBay 公司 EU Analytics 部门负责人

[①] 言外之意，隐私受到威胁。——译者注

译者序

20 年前，IBM 研制的深蓝计算机勉强战胜俄罗斯棋王卡斯帕罗夫，它在体力上的优势似乎比智力方面更明显。但刚刚过去的这一年，谷歌的 AlphaGo 计算机程序打败了围棋高手李世石，它的升级版 Master 威力更是了得，横扫中日韩高手，它擅长走快棋，招法狠毒，令人类高手胆颤。由此可见，近年来，人工智能技术随着硬件、大数据、机器学习技术的发展，取得了长足的进步。

机器学习技术作为人工智能的一个子领域，研究和应用热潮不减，研讨会、学习班和创业项目层出不穷；国内学者入选 AAAI Fellow；该领域的书籍一印再印；人脸识别、自动驾驶、机器翻译、智能客服、物流无人机和家居、医疗、教育机器人等各种应用不断推向市场。从以上种种表现来看，我们处在人工智能时代的风口和前夕。作为该领域的从业者，我们不能满足于看热闹，应努力掌握背后的核心技术——机器学习，力求弄懂该技术，并努力探索其他可能的实现人工智能的方法，把人类智慧的边界向前推进一步。更令人鼓舞的是，大数据产业发展已上升到国家战略层面，我国要实现从数据大国向数据强国的转变，需要一批掌握了数据挖掘、机器学习等相关技术的人才。

本书讲解的是商业网站数据分析和挖掘所用到的机器学习理论和技术。作者先介绍了机器学习的基本概念、Python 机器学习工具栈（NumPy、pandas 和 matplotlib 等），接着分别讲解了无监督和有监督机器学习理论，每种方法都给出形式化描述，其间用到了大量概率统计、线性代数等数学知识，比如最小二乘、相关性、贝叶斯概率和奇异值分解等。作者的统计学背景在这一点上得到了很好的体现。这部分数学知识能够较好地满足有志于深入学习的读者的需要，水平高的读者可以从中感受机器学习模型的数学魅力。介绍完这两大类机器学习理论，作者又从 Web 结构和内容两个方面讲解了 Web 挖掘技术；介绍了信息

检索模型、主题抽取模型 LDA。讲解完机器学习理论和技术之后，作者引入了为 Web 开发完美主义者准备的 Django 框架，让昔日在幕后默默奉献的数据分析高手有机会走到台前，用自己研制的算法驱动一款 Web 产品。作者带领我们利用前面讲解的算法和挖掘技术，用 Django 框架搭建推荐系统和影评分析系统。学到这里，你会不由地感叹 Python 真是全栈工程师的好朋友。数据分析师用 Python 就能从头到尾打造一款智能 Web 产品，可见 Python 的应用范围之广。年初，Facebook 更是开源了 PyTorch 深度学习框架，进一步巩固了 Python 在机器学习领域的地位。

当然，我们最终开发出的产品还比较初级，离最终面向用户的产品在用户体验上还有较大差距，但稍加打磨至少可作为一个最小可行性产品（MVP）先行投入市场，收集用户反馈，日后再图大的改进。此外，限于篇幅，作者也没有讲怎么将系统部署到生产服务器。感兴趣的读者可以试试 Heroku、SAE 等云应用平台，也可以尝试用 Apache、mod_wsgi 在自己的计算机上搭服务器。你可能还需要申请一个域名。这样，你就可以向朋友推荐自己开发的产品了，你具备了向全球用户提供智能 Web 产品的能力！嘿，伙计，快来看，这是我刚刚上线的 Web 推荐系统！用机器学习算法驱动的哦！

感谢人民邮电出版社的陈冀康编辑等为本书编校、出版辛勤付出的各位朋友。读者罗导运行了第 1 章的代码，并指出了原书及译者注中的几处问题。师妹瞿乔阅读了第 2 章译文，她本人也是一本 Python 图书的译者。泰安读者陈新光阅读了第 6 章译文，他正努力学习数据科学知识，祝他学有所成。翻译过程中，我向北京大学冷含莹、东京大学范超、上海健康学院姜萌等朋友请教过问题；我旁听了北大的统计学基础、随机过程等课程，了解了很多统计学概念，参考了市面上现有的多本著作，其中包括大名鼎鼎的西瓜书，查询了 CSDN 等网站的文章，在此一并表示衷心的感谢。感谢西安工业大学的李刚老师、重庆大学杨刘洋同学等读者对翻译工作的支持。最后，感谢我的家人，我翻译图书的时间是用他们的辛勤劳动换来的，因此也更加宝贵。

本人学识有限，且时间仓促，书中翻译错误、不当和疏漏之处在所难免，敬请读者批评指正。

杜春晓

2017 年 2 月

前言

数据科学，尤其是机器学习，成为当下科技商业领域人们热议的议题。这类技术可用来处理用户产生的、数量在不断增长的数据。本书将讲解如何用 Python 语言、Django 框架开发一款 Web 商业应用，还将讲解如何用一些现成的库（sklearn、scipy、NLTK 和 Django 等）处理和分析（通过机器学习技术）应用生成或使用的数据。

本书主要内容

第 1 章，Python 机器学习实践入门，讨论机器学习的主要概念以及数据科学专业人士用 Python 处理数据所使用的几个库。

第 2 章，无监督机器学习，讲解为数据集分簇和从数据中抽取主要特征所用到的算法。

第 3 章，有监督机器学习，讲解预测数据集标签最常用的有监督机器学习算法。

第 4 章，Web 挖掘技术，讨论 Web 数据的组织、分析和从中提取信息的主要技术。

第 5 章，推荐系统，详细介绍当今商业领域所使用的几种最流行的推荐系统。

第 6 章，开始 Django 之旅，介绍开发 Web 应用所用到的 Django 的主要功能和特点。

第 7 章，电影推荐系统 Web 应用，将介绍的机器学习概念付诸实践，动手实现为 Web 用户推荐电影的应用。

第 8 章，影评情感分析应用，再次通过一个实例，使用讲述的知识，分析在线影评的情感倾向和相关性。

本书的阅读前提

读者应该准备一台计算机，装好 Python 2.7，能够运行（和修改）书中各章讲解的代码。

本书的目标读者

任何有一定编程经验（Python）和统计学背景，对机器学习感兴趣和/或希望从事数据科学职业的读者均可从本书受益。

排版约定

本书使用不同的文本样式来区分不同类别的内容。以下是常用样式及其用途说明。

正文中的代码、数据库表名、文件夹名、文件名、文件扩展名、路径名、URL 地址、用户输入的内容和 Twitter 用户名显示方式如下：

“在终端输入以下命令，安装 Django 这个库：sudo pip install django。”

代码块样式如下：

```
INSTALLED_APPS = (
...
'rest_framework',
'rest_framework_swagger',
'nameapp',
)
```

所有的命令行输入或输出使用下面这种样式：

```
python manage.py migrate
```

新的术语和**重要的词语**使用黑体。出现在屏幕上的词语，例如菜单或对话框里，样式如下“如你所见，页面上有两个输入框，输入姓名和邮箱后，单击‘**添加**’，将其添加到数据库”。



此图标表示警告或重要信息。



 此图标表示提示或技巧。 

读者反馈

我们热忱地欢迎读者朋友给予我们反馈，告诉我们你对于这本书的所思所想——你喜欢或是不喜欢哪些内容。大家的反馈对我们来说至关重要，将帮助我们确定到底哪些内容是读者真正需要的。

如果你有一般性建议的话，请发邮件至 feedback@packtpub.com，请在邮件主题中写清书的名称。

如果你是某一方面的专家，对某个主题特别感兴趣，有意向自己或是与别人合作写一本书，请到 www.packtpub.com/authors 查阅我们为作者准备的帮助文档。

客户支持

为自己拥有一本 Packt 出版的书而自豪吧！为了让你的书物有所值，我们还为你准备了以下内容。

下载示例代码

如果你是从 www.packtpub.com 网站购买的图书，用自己的账号登录后，可以下载所有已购图书的示例代码。如果你是从其他地方购买的，请访问 http://www.packtpub.com/support 网站并注册，我们会用邮件把代码文件直接发给你。也可以访问 www.epubit.com.cn 来下载示例代码。

代码文件下载步骤如下。

1．用邮箱和密码登录或注册我们的网站。

2．鼠标移动到页面顶部的 **SUPPORT** 选项卡下。

3．单击 **Code Downloads & Errata**。

4．在搜索框 **Search** 中输入书名。

5．选择你要下载代码文件的图书。

6．从下拉菜单中选择你从何处购买该书。

7．单击 **Code Download** 下载代码文件。

你还可以在 Packt Publishing 网站图书详情页，单击 **Code Files** 按钮下载代码文件。在 **Search** 搜索框中输入书名进行搜索可找到该书的图书详情页。请注意你需要登录网站。

代码下载下来之后，请确保用以下解压工具的最新版本进行解压或抽取文件：

- Windows 用户：WinRAR / 7-Zip；
- Mac 用户：Zipeg / iZip / UnRarX；
- Linux 用户：7-Zip / PeaZip。

本书的代码包在 GitHub 上也存储了一份：https://github.com/PacktPublishing/Machine-Learning-for-the-Web。我们很多其他图书和视频的代码包也存储到了 GitHub 上：https://github.com/PacktPublishing/。将它们检出到本地。

下载本书配套 PDF 文件

我们还为你准备了一个 PDF 文件，该文件包含书中的所有屏幕截图 / 图表。这些彩图能弥补书中黑白图像的不足，有助于你理解本书内容。该文件的下载地址为 http://www.packtpub.com/sites/default/files/downloads/MachineLearningfortheWeb_ColorImages.pdf。

勘误表

即使我们竭尽所能来保证图书内容的正确性，错误也在所难免。如果你在我们出版的任何一本书中发现错误——可能是在文本或代码中——倘若你能告诉我们，我们将会非常感激。你的善举足以减少其他读者在阅读出错位置时的纠结和不快，帮助我们在后续版本中更正错误。如果你发现任何错误，请访问 http://www.packtpub.com/submit-errata，选择相应书籍，单击“Errata Submission Form”链接，输入错误之处的具体信息。你提交的错误得到验证后，我们就会接受你的建议，该处错误信息将会上传到我们网站或是添加到已有勘误表的相应位置。

访问 https://www.packtpub.com/books/content/support，在搜索框中输入书名，可查看该书已有的勘误信息。这部分信息会在 Errata 部分显示。

版权保护

所有媒体在互联网上都面临的一个问题就是侵权。对 Packt 来说，我们严格保护我们

的版权和许可。如果你在网上发现针对我们出版物的任何形式的盗版产品，请立即告知我们地址或网站名称，以便我们进行补救。

请将盗版书籍的网址发送到 copyright@packtpub.com。

如果你能这么做，就是在保护我们的作者，保护我们，只有这样，我们才能继续以优质内容回馈像你这样热心的读者。

问题

你对本书有任何方面的问题，都可以通过 questions@packtpub.com 邮箱联系我们，我们也将尽最大努力来帮你答疑解惑。

作者简介

Andrea Isoni 博士是一名数据科学家、物理学家，他在软件开发领域有着丰富的经验，在机器学习算法和技术方面，拥有广博的知识。此外，他还有多种语言的使用经验，如Python、C/C++、Java、JavaScript、C#、SQL、HTML。他还用过 Hadoop 框架。

译者简介

杜春晓，英语语言文学学士，软件工程硕士。其他译著有《Python 数据挖掘入门与实践》《Python 数据分析实战》和《电子达人——我的第一本 Raspberry Pi 入门手册》等。新浪微博：@宜_生。

技术审稿人简介

Chetan Khatri 是一名数据科学研究员，他共有 4 年半的研究和开发经验。他在 Nazara Technologies Pvt. Ltd 公司担任数据和机器学习方面的首席工程师，主导在游戏和电信订阅业务从事数据科学实践。他曾在一家顶尖的数据公司和印度四大公司其中一家工作，管理数据科学实践平台和后者的资源团队。在这之前，他曾供职于 R & D Lab 和 Eccella Corporation。他拥有印度喀奇大学（KSKV Kachchh University）的计算机科学硕士学位，辅修数据科学，是该学校的金牌得主。

他积极以多种方式为社会做贡献，其中包括为大二学生做讲座，在学术以及其他各种会议上介绍数据科学相关知识，还援助社区一个数据平台。他在学术研究和行业最佳实践两方面均有着相关的专业知识。他喜欢参加数据科学马拉松比赛。他参与发起了 Python 社区——PyKutch。他目前正在探究深层神经网络和增强学习，学习使用并行和分布式计算管理数据。

感谢喀奇大学计算机科学系主任 Devji Chhanga 教授，感谢他引领我走上数据科学研究的正确道路，并给予宝贵的指导意见。同样把感谢送给我亲爱的家人。

Pavan Kumar Kolluru 是一名交叉学科工程师，他是大数据、数字图像和处理、遥感（高光谱数据和图像）方面的专家，精通 Python、R 和 MATLAB 编程。他的研究重点在于如何用机器学习技术、编制算法处理大数据。

他目前正在探索如何找到不同学科之间的联系，以降低数据处理过程在计算和自动化方面的难度。

作为一名数据（图像和信号）处理方面的专业人士和老师，他一直在处理多 / 高光谱数据，该项工作使得他在数据处理、信息抽取和分割方面积累了很多专业知识。他用到的

高级处理技术有 OOA、随机集和马尔可夫随机场。

作为一名程序员和教师，他专注于 Python 和 R 语言，他执教于企业和教育行业的兄弟会。他培训过多批学员，教他们使用 Python 和各种包（信号、图像和数据分析等）。

作为一名机器学习研究员 / 教练，他是分类（有监督和无监督）、建模和数据理解、回归以及数据降维方面的专家。他曾开发出一套大数据（图像或信号）方面的新型机器学习算法，作为他理科硕士阶段的研究成果，该算法将数据降维和分类纳入同一框架，这为他赢得了很高的分数。他培训过多家大型公司的员工，教他们用 Hadoop 和 MapReduce 分析大数据。他的大数据分析专业知识包括 HDFS、Pig、Hive 和 Spark。

Dipanjan Sarkar 是 Intel 公司的一名数据科学家。Intel 是世界上最大的半导体公司，它的使命是让世界更加连通和更具效率。他主要从事分析、商业智能、应用开发和构建大规模的智能系统方面的工作。他从班加罗尔的印度信息技术学院（IIIT）获得信息技术硕士学位。他的专业领域包括软件工程、数据科学、机器学习和文本分析。

Dipanjan 的兴趣包括学习新技术、数据科学和最近的深度学习以及了解具有颠覆性的初创企业动态。业余时间，他喜欢阅读、写作、玩游戏和看情景喜剧。他写过一本关于机器学习的书 *R Machine Learning by Example*，该书由 Packt Publishing 出版。他还为 Packt Publishing 出版的几本机器学习和数据科学图书做过技术评审。

目录

第 1 章 Python 机器学习实践入门

在技术行业，分析和挖掘商业数据的技能正变得越来越重要。公司若有线上业务，可开发利用线上产生的数据，以改进自身业务，或将数据出售给其他公司。重组或分析这些可能具有商业价值的海量信息，只有掌握专业知识的数据科学（或数据挖掘）专业人士才能做得到。数据科学采用机器学习技术将数据转化为模型，以便预测业务领域高度重视的特定实体的行为。这些算法和技术在当今以技术为主导的业务领域是必不可少的。本书讲解这些算法和技术，并介绍如何将其部署到真实的商业环境。你将学到最常用的机器学习技术，并有机会在一系列旨在提高商业智能的练习和应用中使用它们。从本书学到的技能，可用于实际工作。为了充分掌握书中所讨论的各个主题，我们希望你已熟悉 Python 编程语言、线性代数和统计方法。

- 网上有很多关于这些主题的教程和课程，但我们建议你阅读 Python 官方文档（https://docs.python.org/），阅读 A. Bluman 的 *Elementary Statistics* 以及由 G. Casella 和 R. L. Berger 合著的 *Statistical Inference*，理解主要的统计概念和方法。学习线性代数，可阅读 G. Strang 所写的 *Linear Algebra and Its Applications*。

本章作为入门章节，目的是让你熟悉 Python 机器学习的专业人士所使用的更为高级的库和工具，比如 NumPy、pandas 和 matplotlib，帮你掌握必要技术知识，以便实现后续章节的各种技术。讲解本书所用库之前，我们先来阐明机器学习领域的主要概念，并通过一个实例，展示在真实场景中机器学习算法如何给出有用的预测信息。

1.1 机器学习常用概念

本书讨论最常用的机器学习算法，并在练习中加以运用，从而使你熟悉它们。为了解释这些算法，帮你理解本书内容，我们先大体看下几个常用概念，后面会详细介绍。

首先，若要为机器学习下定义，一个较为贴切的定义是，机器学习是计算机科学的一个分支，从模式识别、人工智能和计算学习理论发展而来。我们也可以将机器学习看作是数据挖掘工具，侧重于用数据分析方法理解给定的数据。该学科的目的是，开发能够从先前观测的数据，通过可调整的参数（通常为由双精度数值组成的数组）进行学习的程序，为了改善预测结果，将参数设计为可自动调整的。计算机用这种方式可预测某种行为，概括（generalize）数据的内在结构，而不只是像常见的数据库系统那样对数值进行排序（或检索）。因此，机器学习跟计算统计[①]相关，也是尝试根据先前数据预测某种行为。机器学习方法常见的行业应用有垃圾邮件过滤器、搜索引擎、光学字符识别（OCR）和计算机视觉。既已给出该学科的定义，我们接下来更详细地介绍每种机器学习问题所用术语。

任何学习问题都始于一个包含 n 个样本个体的数据集，未知数据的特性（properties）根据数据集来预测。每个个体通常包含一个以上的数值，因此它是一个向量。向量的组成元素[②]叫作特征（feature）。例如，根据二手车的制造时间、颜色和能耗等车况信息预测其价格。二手车数据集中，每辆车 i 表示成一个特征向量 $x(i)$，对应 i 这辆车的颜色、能耗等车况信息。每辆车 i 还有一个与之对应的目标（或标签）变量 $y(i)$，即二手车的价格。一个训练样例（training example）由一对$(x(i), y(i))$组成。由 N 个数据点组成、用于学习的整个集合叫作训练集$\{(x(i), y(i)); i=1,\cdots,N\}$。符号 x 表示特征（输入）值空间，y 为目标（输出）值空间。为解决问题选用的机器学习算法用数学模型来描述，模型包含一些参数，需在训练集上调试。训练完成后，模型的预测性能用另外两个数据集来评估：验证集和训练集。验证集用来从多个模型中选择能给出最佳结果的那个，测试集通常用来决定所选用模型的实际准确率（precision）[③]。通常，数据集的 50%划作训练集，验证集和测试集则各使用 25%的数据。

学习问题可分为两大类（本书对这两类均有大量介绍）。

无监督学习：给定的训练集只有作为输入的特征向量 x，而未给出任何相对应的标签。该类学习的目标通常为，用聚类算法找出数据中的相似样例，或将数据从高维空间映射（project）到维数更少的空间（盲信号分离算法，比如主成分分析 PCA）。因为每个训练样例通常没有目标值，所以无法直接用训练数据评估模型的错误率；这就需要使用其他方法，评估簇内元素的相似度以及簇间元素的差异程度。这是无监督和有监督学习的一个主要不同点。

① 原文为“computational statics”，其中“statics”应为 statistics。——译者注

② 分量——译者注

③ 精度——译者注

- **有监督学习**[①]：给定训练集的每个个体是一对作为输入的特征向量和标签。该类学习的任务是推断各个参数，预测测试数据的目标值。这些问题可进一步分为：
  - **分类**：数据的目标值属于两个或以上类别，分类的目标是学习如何预测训练集中未标记的数据的类别。分类是一种离散型（与之对应的是连续型）有监督学习方法，标签所代表的类别有限。手写体数字识别是分类问题的实际应用，其目标是将每个特征向量匹配到一组数量有限的离散型类别中的某个类别。
  - **回归**：标签为连续型变量。例如，根据孩子的年龄和体重预测身高就是一个回归问题。

本书第 2 章集中介绍无监督学习方法，第 3 章讨论最常用的有监督学习算法。第 4 章着手讲解 Web 挖掘技术，也可将其看作有监督和无监督方法。第 5 章讲解推荐系统，属于有监督学习范畴。第 6 章介绍 Django Web 框架。第 7 章详细介绍推荐系统（用到 Django 框架和第 5 章相关知识）的实现。我们以一个 Django Web 挖掘应用实例结束本书，实现该应用需使用从第 4 章学到的一些技术。学完本书，你应该能够理解不同的机器学习方法，并有能力将其部署到用 Django 实现的真实 Web 应用。

本章接下来我们将给出一个实例，展示机器学习如何用于实际业务问题，还将给出 Python 库（NumPy、pandas 和 matplotlib）教程。只有掌握这些库的用法，才能实现从后续章节学到的各种算法。

机器学习示例

为了进一步解释机器学习可以拿真实数据做什么，我们一起来看下面这个例子（下面的代码可从作者的 GitHub 主页该书的文件夹下找到，地址为 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_1/。我们从 UCI 机器学习数据库（http://archive.ics.uci.edu/）下载因特网广告数据集（*Internet Advertisements Data Set*，http://archive.ics.uci.edu/ml/datasets/Internet+Advertisements）。这些 Web 广告从各种各样的网页收集而来，每个网页被转换为一个特征向量，其元素为数值类型。从 ad.names 文件，我们可以看到前三个特征表示网页中广告图像的尺寸，其他特征表示图像的 URL 或在文本中出现了哪些特定词语（共有 1558 个特征）。根据网页中是否有广告，标签的取值为 ad 或 nonad。举个例子，一个网页在 ad.data 文件中是这么表示的：

```
125, 125, …, 1. 0, 1, 0, ad.
```

① 还有一类叫作半监督学习，学习器从未标记样本自动学习。见周志华著的《机器学习》。

根据这些数据，一个经典的机器学习任务是找到一个模型，预测哪些网页是广告，哪些不是广告（分类）。首先来看一下包含全部特征向量和标签的 ad.data 文件，我们发现它包含一些用?表示的缺失值。我们可以用 Python 的 pandas 库将?转换为-1（pandas 库详细教程见下节）：

```
import pandas as pd
df = pd.read_csv('ad-dataset/ad.data',header=None)
df=df.replace({'?': np.nan})
df=df.replace({'  ?': np.nan})
df=df.replace({'   ?': np.nan})
df=df.replace({'    ?': np.nan})
df=df.replace({'     ?': np.nan})
df=df.fillna(-1)
```

读入 ad.data 文件中的数据，创建一个 DataFrame 对象，先把每个?替换为一个特殊的元素（replace 函数），然后再替换为-1（fillna 函数）。这样每个标签都被转换为数值类型元素（数据中的其他元素亦同）：

```
adindices = df[df.columns[-1]]== 'ad.'
df.loc[adindices,df.columns[-1]]=1
nonadindices = df[df.columns[-1]]=='nonad.'
df.loc[nonadindices,df.columns[-1]]=0
df[df.columns[-1]]=df[df.columns[-1]].astype(float)
df.apply(lambda x: pd.to_numeric(x))
```

每个 ad.标签转换为 1，而 nonad.则被替换为 0。所有的列（特征）需要是浮点型的数值（用 astype 函数将标签转换为浮点型，在 lambda 函数中用 to_numeric 函数将 df 转换为数值型）。

我们使用 scikit-learn 库（见第 3 章）提供的**支持向量机**（Support Vector Machine，SVM）算法预测数据集中 20%数据的标签。首先，将数据分为两部分：训练集（80%）和测试集（20%）：

```
import numpy as np
dataset = df.values[:,:]
np.random.shuffle(dataset)
data = dataset[:,:-1]
labels = dataset[:,-1].astype(float)
ntrainrows = int(len(data)*.8)
train = data[:ntrainrows,:]
```

```
trainlabels = labels[:ntrainrows]
test = data[ntrainrows:,:]
testlabels = labels[ntrainrows:]
```

上述代码，执行切分数据集之前，用 NumPy 库（教程见下一节）封装的函数，打乱数据的顺序（random.shuffle 函数），确保两个数据集的各行数据是随机选取的。切片操作中的-1，表示数组的最后一列不予考虑。

现在，我们用训练数据训练 SVM 模型：

```
from sklearn.svm import SVC
clf = SVC(gamma=0.001, C=100.)
clf.fit(train, trainlabels)
```

我们声明一个 SVM 模型，指定参数，并将模型赋给 clf 变量。调用 fit 函数，用训练数据拟合（fit）模型（更多内容见第 3 章）。预测 20%测试数据的平均正确率（mean accuracy），用 score 函数来计算，代码如下：

```
score=clf.score(test,testlabels)
print 'score:',score
```

运行上述代码（完整代码见作者 GitHub 主页 chapter_1 文件夹），得到 92%的正确率，也就是说测试集标签的预测结果中，92%的预测标签跟实际标签相同。这就是机器学习的威力所在：根据以往数据，我们能够推断一个网页是否包含广告。为了实现预测功能，我们做了必要的准备工作，用 NumPy 和 pandas 库准备和预处理数据，然后用 scikit-learn 库封装的 SVM 算法处理清洗过的数据。鉴于本书将大量使用 NumPy 和 pandas（有时也会用到 matplotlib）库，下面几节将分别介绍这些库的安装方法以及用它们处理（甚至创建）数据的方法。

安装和导入模块（库）

继续讨论这些库之前，我们先讲怎样在 Python 中安装模块。常用的模块安装方法是，在终端使用 pip 命令：

```
>>> sudo pip install modulename
```
[①]

然后，通常使用下述语句导入模块：

① 注意是在终端而不是 Python shell 里运行 pip 命令。该处代码应去掉前面的 Python shell 提示符。——译者注

```
import numpy as np
```

其中，numpy 是包名，np 指代 numpy，做了这一步操作之后，该库中的所有函数 X 都可以用 np.X 而不必用 numpy.X 访问。本书从此往后假定所有的库（scipy、scikit-learn、pandas、scrapy 和 nltk 等）都用上面这种方式来安装和导入。

1.2　数据的准备、处理和可视化——NumPy、pandas 和 matplotlib 教程

大多数数据在我们拿到时，其形式很不实用，无法直接用机器学习算法处理。如上一个例子所见（上一节），数据中有些元素可能缺失，或某些列不是数值型，因此无法直接用机器学习技术处理。因而，机器学习专家通常花费大量时间清洗和准备数据，转换数据的形式，以便进一步分析或做可视化处理。本节教你用 NumPy 和 pandas 库，用 Python 语言创建、准备和处理数据。matplotlib 小节，将介绍 Python 绘图基础知识。NumPy 教程结合 Python shell 进行讲解，但是代码的 IPython notebook 版和纯 Python 脚本版，都已放到作者 GitHub 主页 chapter_1 文件夹。pandas 和 matplotlib 两个库的讲解则用 IPython notebook。

1.2.1　NumPy 的用法

Numerical Python 或 NumPy 是 Python 的一个开源扩展包，是数据分析和高性能科学计算的基础模块。该库问世后，用 Python 处理大规模、多维数组和矩阵不再是梦想。对于常用数值计算，它提供预先编译好的函数。更进一步来讲，它提供一个巨大的数学函数库来支持数组运算。

该库提供以下功能：

- 用于向量算术运算的快速、多维数组；
- 对数据中所有数组进行快速运算的标准数学函数；
- 线性代数运算；
- 排序、去重（unique）和集合运算；
- 统计和聚合数据。

比起 Python 的标准运算，NumPy 的主要优势在于数组运算速度快。例如，用传统的

求和方法，求 10000000 个元素的和：

```
>>> def sum_trad():
>>>   start = time.time()
>>>   X = range(10000000)
>>>   Y = range(10000000)
>>>   Z = []
>>>   for i in range(len(X)):
>>>       Z.append(X[i] + Y[i])
>>>   return time.time() - start
```

与 NumPy 函数对比：

```
>>> def sum_numpy():
>>>   start = time.time()
>>>   X = np.arange(10000000)
>>>   Y = np.arange(10000000)
>>>   Z=X+Y
>>>   return time.time() - start
>>> print 'time sum:',sum_trad(),' time sum numpy:',sum_numpy()
time sum: 2.1142539978   time sum numpy: 0.0807049274445
```

两种方法所用时间分别为 2.1142539978 和 0.0807049274445。

1. 数组创建

数组对象是 NumPy 库提供的主要功能。数组相当于 Python 的列表（list），但数组所有元素的数值类型相同（通常为浮点型或整型）。借助 array 函数，可用列表定义一个数组对象，需为 array 函数传入两个参数：即将被转换为数组的列表、新生成的数组的类型：

```
>>> arr = np.array([2, 6, 5, 9], float)
>>> arr
array([ 2., 6., 5., 9.])
>>> type(arr)
<type 'numpy.ndarray'>
```

反之，可用如下代码将数组转换为列表：

```
>>> arr = np.array([1, 2, 3], float)
>>> arr.tolist()
[1.0, 2.0, 3.0]
```

```
>>> list(arr)
[1.0, 2.0, 3.0]
```



将数组赋给变量，新建数组，这样做不会在内存为数组创建一个新的副本，它只是将新定义的变量指向原数组对象。

若想用现有数组，创建一个新的数组对象，则要用 copy 函数：

```
>>> arr = np.array([1, 2, 3], float)
>>> arr1 = arr
>>> arr2 = arr.copy()
>>> arr[0] = 0
>>> arr
array([0., 2., 3.])
>>> arr1
array([0., 2., 3.])
>>> arr2
array([1., 2., 3.])
```

此外，还可以用同一个值填充数组，覆盖掉之前的值，得到一个元素全部相同的数组，例如：

```
>>> arr = np.array([10, 20, 33], float)
>>> arr
array([ 10., 20., 33.])
>>> arr.fill(1)
>>> arr
array([ 1., 1., 1.])
```

还可以用 np 的子模块 random 随机选取元素创建数组。例如，将要创建的数组的长度作为 permutation 函数的参数传入，该函数返回一个由整数组成的随机序列[①]：

```
>>> np.random.permutation(3)
array([0, 1, 2])
```

另一种数组创建方法是用 normal 函数从一个正态分布中抽取一列数字：

```
>>> np.random.normal(0,1,5)
```

① 整数的取值范围为大于等于 0，小于指定的参数。——译者注

```
array([-0.66494912, 0.7198794 , -0.29025382, 0.24577752, 0.23736908])
```

参数 0 为正态分布的均值，1 为标准差，5 表示抽取 5 个数字创建数组。若使用均匀分布，random 函数将返回 0 到 1 之间（不包含 0 和 1）的数字：

```
>>> np.random.random(5)
array([ 0.48241564, 0.24382627, 0.25457204, 0.9775729 , 0.61793725])
```

NumPy 还提供几个创建二维数组（矩阵）的函数。例如，identity 函数创建单位矩阵，其维度用参数来指定：

```
>>> np.identity(5, dtype=float)
array([[ 1., 0., 0., 0., 0.],
       [ 0., 1., 0., 0., 0.],
       [ 0., 0., 1., 0., 0.],
       [ 0., 0., 0., 1., 0.],
       [ 0., 0., 0., 0., 1.]])
```

eye 函数返回第 k 条对角线上元素为 1 的矩阵。

```
>>> np.eye(3, k=1, dtype=float)
array([[ 0., 1., 0.],
       [ 0., 0., 1.],
       [ 0., 0., 0.]])
```

创建新数组（1 或 2 维）最常用的函数是 zeros 和 ones，它们按照指定的维度创建数组，并分别用 0 或 1 填充。示例如下：

```
>>> np.ones((2,3), dtype=float)
array([[ 1., 1., 1.],
       [ 1., 1., 1.]])
>>> np.zeros(6, dtype=int)
array([0, 0, 0, 0, 0, 0])
```

而 zeros_like 和 ones_like 函数，创建的是跟现有数组元素的类型[①]和维度都相同的数组：

```
>>> arr = np.array([[13, 32, 31], [64, 25, 76]], float)
>>> np.zeros_like(arr)
array([[ 0., 0., 0.],
```

① 指数组内各元素的数值类型。——译者注

```
       [ 0., 0., 0.]])
>>> np.ones_like(arr)
array([[ 1., 1., 1.],
       [ 1., 1., 1.]])
```

另外一种创建二维数组的方法是，用 vstack 函数（垂直方向合并）合并一维数组：

```
>>> arr1 = np.array([1,3,2])
>>> arr2 = np.array([3,4,6])
>>> np.vstack([arr1,arr2])
array([[1, 3, 2],
      [3, 4, 6]])
```

二维数组也可以用 random 子模块按照某种分布进行创建。例如，随机从 0 到 1 的均匀分布中选取数字作为数组元素，创建一个 2×3 型数组，方法如下：

```
>>> np.random.rand(2,3)
array([[ 0.36152029, 0.10663414, 0.64622729],
    [ 0.49498724, 0.59443518, 0.31257493]])
```

另一种经常用来创建数组的分布是多元正态分布：

```
>>> np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]], size=[5,])
array([[ 11.8696466 , -0.99505689],
       [ 10.50905208, 1.47187705],
       [ 9.55350138, 0.48654548],
       [ 10.35759256, -3.72591054],
       [ 11.31376171, 2.15576512]])
```

列表[10,0]是均值向量，[[3, 1], [1, 4]]是协方差矩阵，5 是要抽取的元素数量。

表 1.1

方法	用途
tolist	将 NumPy 数组转换为 Python 列表的函数
copy	复制 NumPy 数组元素的函数
ones, zeros	创建用 1 或 0 填充的数组的函数
zeros_like, ones_like	该函数用来创建与作为参数的列表形状相同的二维数组
fill	将数组元素替换为某一特定元素的函数

续表

方法	用途
identity	创建单位矩阵的函数
eye	该函数用来创建第 k 条对角线上元素为 0 的矩阵
vstack	将数组合并为二维数组的函数
random 子模块：random、permutation、normal、rand、multivariate_normal 等	random 子模块从某种分布抽取元素，创建数组

2．数组操作

访问列表元素、切片以及其他 Python 列表的所有常见操作，均能以相同或相似的方式作用于数组：

```
>>> arr = np.array([2., 6., 5., 5.])
>>> arr[:3]
array([ 2., 6., 5.])
>>> arr[3]
5.0
>>> arr[0] = 5.
>>> arr
array([ 5., 6., 5., 5.])
```

数组所包含的不同元素也可以获取到，用 unique 函数即可：

```
>>> np.unique(arr)
array([ 5., 6.])
```

数组元素的排序也可以用 sort 函数。数组的索引用 argsort 函数获取：

```
>>> arr = np.array([2., 6., 5., 5.])
>>> np.sort(arr)
array([ 2., 5., 5., 6.])
>>> np.argsort(arr)
array([0, 2, 3, 1])
```

用 shuffle 函数也可以调整数组元素，使其随机排列：

```
>>> np.random.shuffle(arr)
>>> arr
```

```
array([ 2., 5., 6., 5.])
```

NumPy 数组类似，也有一个内置函数 array_equal，用来比较两个数组是否相等[①]：

```
>>> np.array_equal(arr,np.array([1,3,2]))
False
```

然而，多维数组与列表操作不同。事实上，多维列表各维度用逗号分隔的形式依次指定（而列表用方括号[②]）。例如，二维列表（矩阵）元素访问方法如下：

```
>>> matrix = np.array([[ 4., 5., 6.], [2, 3, 6]], float)
>>> matrix
array([[ 4., 5., 6.],
       [ 2., 3., 6.]])
>>> matrix[0,0]
4.0
>>> matrix[0,2]
6.0
```

对数组的各维进行切片操作使用英文冒号:，冒号前后为位于起始位置和结束位置的元素的索引：

```
>>> arr = np.array([[ 4., 5., 6.], [ 2., 3., 6.]], float)
>>> arr[1:2,2:3]
array([[ 6.]])
```

仅用冒号:，不用数字，表示冒号所在轴上的所有元素都在切片范围之内：

```
>>> arr[1,:]
array([2, 3, 6])
>>> arr[:,2]
array([ 6., 6.])
>>> arr[-1:,-2:]
array([[ 3., 6.]])
```

flatten 函数可将多维数组变为一维数组：

---

① 相等，指形状和元素是否均相等。此外，列表比较用 cmp 函数。——译者注

② >>>multilist = [[4, 5, 6], [2, 3, 6]]
   >>>multilist[0][0]
   4 ——译者注

```
>>> arr = np.array([[10, 29, 23], [24, 25, 46]], float)
>>> arr
array([[ 10., 29., 23.],
       [ 24., 25., 46.]])
>>> arr.flatten()
array([ 10., 29., 23., 24., 25., 46.])
```

我们还可以查看数组对象，获取相关信息。用 shape 属性，可得到数组的大小：

```
>>> arr.shape
(2, 3)
```

该例中，arr 是一个 2 行 3 列的矩阵。dtype 属性返回数组元素的类型：

```
>>> arr.dtype
dtype('float64')
```

数值类型 float64 用来存储双精度（8 字节）实数（类似于 Python 的标准 float 类型）。其他数据类型有 int64、int32 和字符串。数组的数据类型可以转换。例如：

```
>>> int_arr = matrix.astype(np.int32)
>>> int_arr.dtype
dtype('int32')
```

len 函数返回数组第一维的长度：

```
>>> arr = np.array([[ 4., 5., 6.], [ 2., 3., 6.]], float)
>>> len(arr)
2
```

关键字 in，类似于它在 Python for 循环中的用法，可用来判断数组是否包含某个元素：

```
>>> arr = np.array([[ 4., 5., 6.], [ 2., 3., 6.]], float)
>>> 2 in arr
True
>>> 0 in arr
False
```

reshape 函数可调整数组的维度。[①]例如，8 行 1 列的矩阵可调整为 4 行 2 列的矩阵：

---

① 直译就是 reshape 函数可调整元素所在的维度。——译者注

```
>>> arr = np.array(range(8), float)
>>> arr
array([ 0., 1., 2., 3., 4., 5., 6., 7.])
>>> arr = arr.reshape((4,2))
>>> arr
array([[ 0., 1.],
       [ 2., 3.],
       [ 4., 5.],
       [ 6., 7.]])
>>> arr.shape
(4, 2)
```

此外，还支持矩阵的转置运算；也就是说，用 transpose 函数可互换两个维度，创建一个新数组：

```
>>> arr = np.array(range(6), float).reshape((2, 3))
>>> arr
array([[ 0., 1., 2.],
       [ 3., 4., 5.]])
>>> arr.transpose()
array([[ 0., 3.],
       [ 1., 4.],
       [ 2., 5.]])
```

数组还可以用 T 属性实现转置：

```
>>> matrix = np.arange(15).reshape((3, 5))
>>> matrix
array([[ 0, 1, 2, 3, 4],
       [ 5, 6, 7, 8, 9],
       [10, 11, 12, 13, 14]])
>>>matrix .T
array([[ 0, 5, 10],
       [ 1, 6, 11],
       [ 2, 7, 12],
       [ 3, 8, 13],
       [ 4, 9, 14]])
```

另一种调整数组元素位置的方法是，用 newaxis 函数增加维度：

```
>>> arr = np.array([14, 32, 13], float)
>>> arr
```

```
array([ 14., 32., 13.])
>> arr[:,np.newaxis]
array([[ 14.],
       [ 32.],
       [ 13.]])
>>> arr[:,np.newaxis].shape
(3,1)
>>> arr[np.newaxis,:]
array([[ 14., 32., 13.]])
>>> arr[np.newaxis,:].shape
(1,3)
```

上述例子，两个新数组都是二维的。由 newaxis 生成的第二个数组长度为 1。

NumPy 数组的连接操作用 concatenate 函数，句法形式取决于数组的维度。多个一维数组可相继连接，将要连接的多个数组置于元组中作为参数传入即可：

```
>>> arr1 = np.array([10,22], float)
>>> arr2 = np.array([31,43,54,61], float)
>>> arr3 = np.array([71,82,29], float)
>>> np.concatenate((arr1, arr2, arr3))
array([ 10., 22., 31., 43., 54., 61., 71., 82., 29.])
```

多维数组必须指定沿哪条轴连接。否则，NumPy 默认沿第一条轴连接：

```
>>> arr1 = np.array([[11, 12], [32, 42]], float)
>>> arr2 = np.array([[54, 26], [27,28]], float)
>>> np.concatenate((arr1,arr2))
array([[ 11., 12.],
       [ 32., 42.],
       [ 54., 26.],
       [ 27., 28.]])
>>> np.concatenate((arr1,arr2), axis=0)
array([[ 11., 12.],
       [ 32., 42.],
       [ 54., 26.],
       [ 27., 28.]])
>>> np.concatenate((arr1,arr2), axis=1)
array([[ 11., 12., 54., 26.],
       [ 32., 42., 27., 28.]])
```

将大量数据保存为二进制文件而不用原格式存储，这样的情况很常见。NumPy 的 tostring 函数可将数组转化为二进制字符串。当然，这个过程是可逆的，fromstring 函数可将二进制

字符串还原为数组。例如：

```
>>> arr = np.array([10, 20, 30], float)
>>> str = arr.tostring()
>>> str
'\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x004@\x00\x00\x00\x00\
x00\x00>@'
>>> np.fromstring(str)
array([ 10., 20., 30.])
```

表 1.2

方法	用途
unique	从数组选取所有不同元素的函数
random、shuffle	调整数组元素位置，使其随机排列的函数
sort、argsort	sort 按照升序排列数组元素 argsort 返回数组元素升序排列时的索引列表[①]
array_equal	比较两个数组，如果相同，返回 True（否则返回 False）
flatten	将二维数组转换为一维数组
transpose	计算二维数组的转置
reshape	调整二维数组的元素，改变数组的形状
concatenate	沿现有的轴，连接多个数组[②]
fromstring 、tostring	二进制字符串和数组之间的互相转换

3．数组运算

NumPy 数组显然支持常见的数学运算。例如：

```
>>> arr1 = np.array([1,2,3], float)
>>> arr2 = np.array([1,2,3], float)
>>> arr1 + arr2
array([2.,4., 6.])
>>> arr1-arr2
```

① 返回结果的类型为 NumPy 数组。——译者注

② 原文为“Concatenate two -dimensional arrays into one matrix”，意思是拼接两个数组，形成一个矩阵，不够全面。因此，此处译文是根据 SciPy.org 文档对 concatenate 函数的描述翻译的。——译者注

```
array([0., 0., 0.])
>>> arr1 * arr2
array([1, 4., 9.])
>>> arr2 / arr1
array([1., 1., 1.])
>>> arr1 % arr2
array([0., 0., 0.])
>>> arr2**arr1
array([1., 4., 27.])
```

既然上述运算都作用于元素级别，这就要求参与运算的数组大小相同。如果该条件不能满足，就会返回错误：

```
>>> arr1 = np.array([1,2,3], float)
>>> arr2 = np.array([1,2], float)
>>> arr1 + arr2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: shape mismatch: objects cannot be broadcast to a single shape
```

上述错误信息的意思是无法对对象进行广播（broadcast），因为大小不同的数组参与运算的唯一方法叫作广播。广播的意思是数组维度不同时，维度少的数组将多次重复自身，直到它跟另外一个数组维度相同。看下面的示例：

```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> arr2 = np.array([1, 2], float)
>>> arr1
array([[ 1., 2.],
       [ 3., 4.],
       [ 5., 6.]])
>>> arr2
array([1., 2.])
>>> arr1 + arr2
array([[ 2., 4.],
       [ 4., 6.],
       [ 6., 8.]])
```

arr2 被广播成大小跟 arr1 相同的二维数组。因而，对于 arr1 的每一维，arr2 都重复自身一次，相当于 arr2 变换为下面这个数组后再参加运算：

```
array([[1., 2.],[1., 2.],[1., 2.]])
```

如要明确指定数组的广播方式，可用 newaxis 常量[①]：

```
>>> arr1 = np.zeros((2,2), float)
>>> arr2 = np.array([1., 2.], float)
>>> arr1
array([[ 0., 0.],[ 0., 0.]])
>>> arr2
array([1., 2.])
>>> arr1 + arr2
array([[-1., 3.],[-1., 3.]])
>>> arr1 + arr2[np.newaxis,:]
array([[1., 2.],[1., 2.]])
>>> arr1 + arr2[:,np.newaxis]
array([[1.,1.],[ 2., 2.]])
```

跟 Python 列表[②]不同的是，数组支持按条件查询，用布尔数组过滤元素就是一个典型的例子：

```
>>> arr = np.array([[1, 2], [5, 9]], float)
>>> arr >= 7
array([[ False, False],
[False, True]], dtype=bool)
>>> arr[arr >= 7]
array([ 9.])
```

可以用多个布尔表达式获取数组的子集：

```
>>> arr[np.logical_and(arr > 5, arr < 11)]
>>> arr
array([ 9.])
```

我们可以根据索引选取元素，用目标元素的索引构造一个数据类型为整型的数组，然后，将索引数组放到目标数组的后面，并用方括号括起来。例如：

```
>>> arr1 = np.array([1, 4, 5, 9], float)
>>> arr2 = np.array([0, 1, 1, 3, 1, 1, 1], int)
```

① 代码中 arr1+arr2 输出结果有误。应为：

```
arr1 + arr2

array([[ 1.,  2.],
       [ 1.,  2.]])
```

——译者注

② 要实现 Python 列表筛选操作，可使用内置函数 filter。——译者注

```
>>> arr1[arr2]
array([ 1., 4., 4., 9., 4., 4., 4.])
```

上述第 3 行代码，表示按照 arr2 指定的索引顺序，从数组 arr1 中选取相应的元素，也就是选取 arr1 的第 0、第 1、第 1、第 3、第 1、第 1 和第 1 个元素。用列表存储目标元素的索引，可以达到同样的选取效果：

```
>>> arr = np.array([1, 4, 5, 9], float)
>>> arr[[0, 1, 1, 3, 1]]
array([ 1., 4., 4., 9., 4.])
```

多维数组的选取操作，需要使用多个一维度的索引数组，每个维度对应一个索引数组。索引数组放到 Python 列表中[①]，再将 Python 列表置于多维数组后面的方括号之中。

第 1 个种子数组（selection array）存储矩阵元素的行号，第 2 个种子数组存储列号。例如：

```
>>> arr1 = np.array([[1, 2], [5, 13]], float)
>>> arr2 = np.array([1, 0, 0, 1], int)
>>> arr3 = np.array([1, 1, 0, 1], int)
>>> arr1[arr2,arr3]
array([ 13., 2., 1., 13.])
```

arr2 的元素为 arr1 元素的行号，而 arr3 的元素则为 arr1 元素的列号，因此从 arr1 选取的第 1 个元素为位于第 1 行、第 1 列的 13。

take 函数支持以索引数组为参数，从调用它的数组选取元素，效果等同于上面的方括号选择法：

```
>>> arr1 = np.array([7, 6, 6, 9], float)
>>> arr2 = np.array([1, 0, 1, 3, 3, 1], int)
>>> arr1.take(arr2)
array([ 6., 7., 6., 9., 9., 6.])
```

用 axis 参数指定维度，take 函数可从调用它的多维数组、沿指定维度选取一部分元素：

```
>>> arr1 = np.array([[10, 21], [62, 33]], float)
>>> arr2 = np.array([0, 0, 1], int)
>>> arr1.take(arr2, axis=0)
```

① 指示例代码第 4 行中的［arr2, arr3］。——译者注

```
array([[ 10., 21.],
       [ 10., 21.],
       [ 62., 33.]])
>>> arr1.take(arr2, axis=1)
array([[ 10., 10., 21.],
       [ 62., 62., 33.]])
```

put 函数为 take 函数的逆操作，它有两个参数：将元素放到什么位置（索引列表）、被投放的元素来自哪个数组。put 函数将一个数组的元素放到调用该函数的另一个数组的指定位置：

```
>>> arr1 = np.array([2, 1, 6, 2, 1, 9], float)
>>> arr2 = np.array([3, 10, 2], float)
>>> arr1.put([1, 4], arr2)
>>> arr1
array([ 2., 3., 6., 2., 10., 9.])
```

本节最后，我们想提醒你注意，二维数组的乘法运算也是元素级的（但矩阵乘法不是）：

```
>>> arr1 = np.array([[11,22], [23,14]], float)
>>> arr2 = np.array([[25,30], [13,33]], float)
>>> arr1 * arr2
array([[ 275., 660.],
       [ 299., 462.]])
```

表 1.3

方法	用途
take	以一个整数数组做参数表示索引，从另一个数组选取相应的元素
put	将一个数组给定位置的元素替换为另一个数组的元素

4. 线性代数运算

矩阵之间最常用的运算是，矩阵与其转置矩阵的内积 X^TX。计算内积，用 np.dot 函数[①]：

---

① np.dot(X.T, X)的输出结果附图有误，应为：

```
np.dot(X .T, X)
```

```
array([[125, 140, 155, 170, 185],
       [140, 158, 176, 194, 212],
       [155, 176, 197, 218, 239],
       [170, 194, 218, 242, 266],
       [185, 212, 239, 266, 293]])
```

——译者注

```
>>> X = np.arange(15).reshape((3, 5))
>>> X
array([[ 0, 1, 2, 3, 4],
       [ 5, 6, 7, 8, 9],
       [10, 11, 12, 13, 14]])
>>> X.T
array([[ 0, 5, 10],
       [ 1, 6, 11],
       [ 2, 6, 12],
       [ 3, 8, 13],
       [ 4, 9, 14]])
>>>np.dot(X .T, X)#X^T X
array([[ 2.584 , 1.8753, 0.8888],
       [ 1.8753, 6.6636, 0.3884],
       [ 0.8888, 0.3884, 3.9781]])
```

有几个函数可直接计算数组（矩阵或向量）不同类型的积（内积、外积、向量积）。

一维数组（向量）的内积与点积相同：

```
>>> arr1 = np.array([12, 43, 10], float)
>>> arr2 = np.array([21, 42, 14], float)
>>> np.outer(arr1, arr2)
array([[ 252., 504., 168.],
       [ 903., 1806., 602.],
       [ 210., 420., 140.]])
>>> np.inner(arr1, arr2)
2198.0
>>> np.cross(arr1, arr2)
array([ 182., 42., -399.])
```

NumPy 的 linalg 子模块，实现了矩阵的多种线性代数运算。例如，计算矩阵的行列式的值：

```
>>> matrix = np.array([[74, 22, 10], [92, 31, 17], [21, 22, 12]], float)
>>> matrix
array([[ 74., 22., 10.],
       [ 92., 31., 17.],
       [ 21., 22., 12.]])
>>> np.linalg.det(matrix)
-2852.0000000000032
```

inv 函数生成矩阵的逆矩阵[①]：

```
>>> inv_matrix = np.linalg.inv(matrix)
>>> inv_matrix
array([[ 0.00070126, 0.01542777, -0.02244039],
       [ 0.26192146, -0.23772791, 0.11851332],
       [-0.48141655, 0.4088359 , -0.09467041]])
>>> np.dot(inv_matrix,matrix)
array([[ 1.00000000e+00, 2.22044605e-16, 4.77048956e-17],
       [ -2.22044605e-15, 1.00000000e+00, 0.00000000e+00],
       [ -3.33066907e-15, -4.44089210e-16, 1.00000000e+00]])
```

矩阵的特征值（eigenvalues）和特征向量（eigenvectors）计算方法很简单：

```
>>> vals, vecs = np.linalg.eig(matrix)
>>> vals
array([ 107.99587441, 11.33411853, -2.32999294])
>>> vecs
array([[-0.57891525, -0.21517959, 0.06319955],
       [-0.75804695, 0.17632618, -0.58635713],
       [-0.30036971, 0.96052424, 0.80758352]])
```

表 1.4

方法	用途
dot	两个数组的点积
inner	两个多维数组的内积
linalg 模块中的 linalg.det、linalg.inv 和 linalg.eig 等函数	linalg 模块包括多个线性代数运算方法，其中有求矩阵的行列式的值（det）、矩阵的逆（inv）以及矩阵的特征值和特征向量（eig）

5．统计和数学函数

NumPy 提供一组计算数组元素统计信息的函数。聚合型运算，比如求和、均值、中位数和标准差，可通过访问数组的相应属性得到。例如，随机选取元素（服从某正态分布），创建一个数组，我们可以用以下两种方法计算数组元素的均值：

① np.dot(inv_matrix, matrix)的输出结果附图有误，应为：

```
np.dot(inv_matrix, matrix)

array([[  1.00000000e+00,  -1.11022302e-16,  -1.11022302e-16],
       [  1.77635684e-15,   1.00000000e+00,  -4.44089210e-16],
       [ -3.33066907e-15,  -4.44089210e-16,   1.00000000e+00]])
```

——译者注

```
>>> arr = np.random.rand(8, 4)
>>> arr.mean()
0.45808075801881332
>>> np.mean(arr)
0.45808075801881332
>>> arr.sum()
14.658584256602026
```

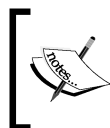
所有这一类函数见表 1.5。

表 1.5

方法	用途
mean	各元素的均值。空数组，均值默认为 NaN
std、var	计算数组的标准差（std）和方差（var）。可指定自由度参数（默认为数组的长度）
min、max	求数组最小值（min）和最大值（max）
argmin、argmax	返回最小（argmin）和最大（argmax）元素的索引

1.2.2 理解 pandas 模块

Python 的 pandas 模块功能强大，包含大量用于分析数据结构的函数。它依赖于 NumPy 库。pandas 的设计初衷是降低数据分析操作的难度，提升速度。比起 Python 的标准函数，pandas 函数性能更高，尤其擅长文件读写、数据库操作；pandas 是数据处理的最佳选择。探索数据所包含的信息，主要方法有哪些以及如何用 pandas 进行操作，下面几节将给出答案。我们先讲解数据在 pandas 中的存储形式和数据的加载方法。



本书从此往后，我们都用如下语句导入 pandas:

```
import pandas as pd
```

因此，下文所有代码，只要有 pd，均指 pandas。

1．探索数据

我们先介绍 pandas 只有一维的数组类对象 Series，以此引入 pandas 的数据库结构 **DataFrame**。Series 可以存储 NumPy 所有类型的数据，同时还存储数据的标签——索引。我们来看一个简单的例子：

```
In [8]: obj = pd.Series([3,5,-2,1])
        obj

Out[8]: 0    3
        1    5
        2   -2
        3    1
        dtype: int64
```

obj 对象由两类值组成，右侧为元素，左侧为元素对应的索引。给定元素数组的长度为 N，索引则默认从 0 排到 N-1。Series 的元素数组和索引对象，可分别用 values 和 index 属性获取：

```
In [9]: obj.values

Out[9]: array([ 3,  5, -2,  1])

In [10]: obj.index

Out[10]: Int64Index([0, 1, 2, 3], dtype='int64')
```

NumPy 数组运算，索引会予以保留（例如标量乘法、用布尔数组过滤或用数学函数处理数组）：

```
In [11]: obj *2

Out[11]: 0     6
         1    10
         2    -4
         3     2
         dtype: int64

In [12]: obj[obj>2]

Out[12]: 0    3
         1    5
         dtype: int64
```

Python 字典可转换为 Series 对象，但是字典的键被转换为索引：

```
In [19]: data = {'a': 30, 'b': 70, 'c': 160, 'd': 5}
         obj = pd.Series(data)
         obj

Out[19]: a     30
         b     70
         c    160
         d      5
         dtype: int64
```

也可以用单独的列表作为索引：

```
In [20]: index = ['a','b','c','d','g']
         obj = pd.Series(data, index=index)
         obj

Out[20]: a     30
         b     70
         c    160
         d      5
         g    NaN
         dtype: float64
```

上述例子，最后一个索引 g，没有对应的元素，因此 pandas 默认插入 **NaN**（Not a Number，非数字）。

我们用缺失值或 NAN 来指代缺失的数据。用 pandas 的 isnull 和 notnull 函数，可找出缺失值：

```
In [16]: pd.isnull(obj)

Out[16]: a    False
         b    False
         c    False
         d    False
         dtype: bool

In [17]: pd.notnull(obj)

Out[17]: a    True
         b    True
         c    True
         d    True
         dtype: bool
```

现在，我们可以从一个 CSV 文件导入数据到 DataFrame 结构。DataFrame 这种数据结构，有一组按顺序排列的列，每一列可以是不同的数据类型（数值、字符串、布尔值等）。DataFrame 有两种索引（行、列索引）。我们也可以将 DataFrame 看成一个由 Series 对象组成的字典，每个 Series 里面，所有元素的索引相同（列的标题）。下面我们结合 ad.data 文件中的数据进行讲解，该数据可从 http://archive.ics.uci.edu/ml/datasets/Internet+Advertisements 下载。在前面机器学习的例子已介绍过。

在终端输入以下代码导入数据（该例中，数据文件的路径为 data_example/ad-dataset/ad.data[①]）：

```
In [4]: data = pd.read_csv("data_example/ad-dataset/ad.data",header=None)
```

该文件没有标题行（故将 header 参数设置为 none），因此使用数字作为各列的名称。

① 原书此处为“ad-data”，文件名实际为“ad.data”。——译者注

在 data 对象上调用 describe 函数，可得到 DataFrame 的各种总描述性统计信息：

```
In [5]: data.describe()
Out[5]:
```

	4	5	6	7	8	9	10	11	12	13	...
count	3279.000000	3279.000000	3279.000000	3279.000000	3279.000000	3279.000000	3279.000000	3279.000000	3279.000000	3279.000000	...
mean	0.004270	0.011589	0.004575	0.003355	0.003965	0.011589	0.003355	0.004880	0.009149	0.004575	...
std	0.065212	0.107042	0.067491	0.057831	0.062850	0.107042	0.057831	0.069694	0.095227	0.067491	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

8 rows × 1554 columns

上面总结了几种定量信息。由此可见，该数据集总共有 1554 个数值类型的列（因为没有标题行，列的名称用数字表示）、3279 行（对每一列调用 count 函数）。每一列都有一组统计指标（均值、标准差、最小值、最大值和分位数），这些统计数据有助于我们对 DataFrame 中数据的定量信息做出初步估计。

用 columns 属性可获取到所有列的名称：

```
In [25]: data.columns
Out[25]: Int64Index([    0,     1,     2,     3,     4,     5,     6,     7,     8,     9,
                    ...
                    1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558],
                   dtype='int64', length=1559)
```

所有列的名称为 int64 类型，下述命令返回所有列的实际数据类型：

```
In [26]: data.dtypes
Out[26]: 0        object
         1        object
         2        object
         3        object
         4         int64
         5         int64
                   ...
         1557      int64
         1558     object
         dtype: object
```

前 4 列和标签列（最后 1 列）为 object 类型，其余为 int64 类型。列的访问方法有两种。第 1 种，指定列的名称，与指定字典的键相似：

```
In [6]: data[1]
Out[6]: 0        125
        1        468
        29       234
                ...
        3277       ?
        3278      40
        Name: 1, dtype: object
```

用列表形式，指定多个列名称，可获取到多列：

```
In [27]: data[[1,20]]
Out[27]:
```

	1	20
0	125	0
1	468	0
2	230	0
3	468	0
4	468	0
...	...	...
3277	?	0
3278	40	0

3279 rows × 2 columns

另一种访问列的方法是点号句法，这只有在列名称也是 Python 变量（中间没有空格），没有重名的 DataFrame 属性或函数（比如 count 或 sum），并且列名称还必须是字符串类型时，才能使用该方法（该例中，列名称为 int64 类型，因此不能用此方法）。

若想对 DataFrame 中的内容有一个大致的了解，可使用 head()函数。它默认返回一列的前 5 个元素（或 DataFrame 的前 5 行）：

```
In [28]: data[1].head()
Out[28]: 0      125
         1      468
         2      230
         3      468
         4      468
         Name: 1, dtype: object
```

当然也可以用 tail()函数，它默认返回最后 5 个元素或 5 行。在 head()或 tail()函数中指定数字 *n*，将返回所选列的前、后 *n* 个元素：

```
In [29]: data[1].head(10)
Out[29]: 0      125
         1      468
         2      230
         3      468
         4      468
         5      468
         6      460
         7      234
         8      468
         9      468
         Name: 1, dtype: object
```

用 Python 的标准切片句法，也可以从 DataFrame 中获取到一定数量的行：

```
In [7]: data[1:3]
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.

2 rows × 1559 columns

上述代码仅获取到 DataFrame 的前两行（当然还有标题）。

2．操作数据

行的选取方法有多种，比如指定索引或按条件选取：

```
In [31]: data[data[1]> 0].head(4)
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	...	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558
0	125	125	1.0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
3	60	468	7.8	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.

4 rows × 1559 columns

或者，按多个条件选取数据：

```
In [32]: data[(data[1]> 0) & (data[1558]=='ad.')].head(4)
```

Out[32]:

	0	1	2	3	4	5	6	7	8	9	...	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558
0	125	125	1.0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
3	60	468	7.8	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.

4 rows × 1559 columns

上面返回的数据为特征 1 大于 0 且包含广告的网页。

ix 方法通过指定索引来选择相应的行：

```
In [33]: data.ix[:3]
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558
0	125	125	1.0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
3	60	468	7.8	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.

4 rows × 1559 columns

此外，也可以使用 iloc 函数：

```
In [34]: data.iloc[:3]
Out[34]:
```

	0	1	2	3	4	5	6	7	8	9	...	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558
0	125	125	1.0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.

3 rows × 1559 columns

ix 和 iloc 的不同点在于，ix 操作的是索引列标签的名称，而 iloc 操作的是索引的位置（因此它只能接收整数）。因此，上述例子，ix 一直找到标签 3 出现为止（共 4 行），而 iloc 函数返回 DataFrame 的前 3 行。访问 DataFrame 内部数据，还有一个函数叫作 loc，它查找索引列的标签名，返回相应的行[①]。例如：

```
In [35]: data.loc[:3]
Out[35]:
```

	0	1	2	3	4	5	6	7	8	9	...	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558
0	125	125	1.0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.
3	60	468	7.8	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	ad.

4 rows × 1559 columns

注意该函数与 Python 的标准切片方法不同，因为结果包括起始和结束位置的行（该例中，输出结果包含索引为 3 的行）。

再来看其他操作。DataFrame 对象的一整列可设置为同一个值：

```
In [36]: data[1547] = 0
```

也可以将指定的单元格设置为我们想要的值：

```
In [37]: data.ix[3,1]=0
```

或将整行设置为一组值（该例使用随机数 0 或 1 和 ad.标签）。

```
In [38]: import random
         data.ix[0] = [random.randint(0,1) for r in xrange(1558)]+['ad.']
```

数组转换为 Series 对象后，可作为新的一行追加到 DataFrame 的末尾：

```
In [40]: row = [random.randint(0,1) for r in xrange(1558)]+['ad.']
         data = data.append(pd.Series(row,index = data.columns),ignore_index=True)
```

用 loc 函数可在 DataFrame 最后增加一行：

① loc 方法，如果是切片操作，起始和结束标签所对应的行连同它们之间的行都能获取到。如果是 loc[label]，则只能获取到标签为 label 的行。label 既可以是数字也可以是字符串。——译者注

```
In [70]: data.loc[len(data)] = row
```

追加列则非常简单，将元素赋给 DataFrame 新的列即可：

```
In [41]: data['newcolumn'] = 'test value'
         data.columns

Out[41]: Index([           0,            1,            2,            3,            4,
                           5,            6,            7,            8,            9,
                ...
                        1550,         1551,         1552,         1553,         1554,
                        1555,         1556,         1557,         1558, u'newcolumn'],
               dtype='object', length=1560)
```

上述例子，新增的列所有元素的值为 *test value*。删除列，可用 drop 函数：

```
In [56]: data = data.drop('newcolumn', 1)
         data.columns

Out[56]: Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
                ...
                1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558],
               dtype='object', length=1559)
```

出于多种原因，数据集也许包含重复数据。pandas 的 duplicated 方法可判断每一行是否是对其他行的重复：

```
In [42]: data.duplicated()

Out[42]: 0       False
         1       False
         2       False
         3       False
         4       False
                  ...
         3279    False
         dtype: bool
```

drop_duplicates 函数比 duplicated 函数更强大，它返回的 DataFrame 仅包含去重过后剩余的所有元素。例如，使用该函数，我们可以找出标签这一列两个不同的元素是：

```
In [43]: data[1558].drop_duplicates()

Out[43]: 0         ad.
         459    nonad.
         Name: 1558, dtype: object
```

我们还可以方便地将上述结果转换为列表：

```
In [44]: data[1558].drop_duplicates().tolist()

Out[44]: ['ad.', 'nonad.']
```

我们可以把标签转换为数值，前面机器学习示例一节讲过这种方法：

```
In [46]: adindices = data[data .columns[-1]]== 'ad.'
         data.loc[adindices,data .columns[-1]]=1
         nonadindices = data[data .columns[-1]]=='nonad.'
         data.loc[nonadindices,data .columns[-1]]=0
```

标签列仍然为 object 类型：

```
In [47]: data[1558].dtypes
Out[47]: dtype('O')
```

然后，我们可以将标签列转换为浮点型：

```
In [63]: data[data.columns[-1]]=data[data.columns[-1]].astype(float)
```

前 4 列包含不同类型的数据（字符串、?和浮点型数字）。我们删除字符串类型的元素后，才能将各列元素转换为数值型。我们可以用 replace 函数将所有的?实例（缺失值）替换为 NaN：

```
In [71]: data=data.replace({'?': np.nan})
         data=data.replace({'   ?': np.nan})
         data=data.replace({'    ?': np.nan})
         data=data.replace({'     ?': np.nan})
         data=data.replace({'      ?': np.nan})
```

现在，我们可以用两种方法处理包含缺失值的行。方法一，用 dropna 方法直接删除包含缺失值的行：

```
In [73]: data=data.dropna()
```

方法二，包含缺失数据的行，除了直接将其删除（可能删除重要信息）外，也可为其填充数据。用 fillna 方法，向这些空的单元格填充一个常量，可满足大多数需求：

```
In [74]: data=data.fillna(-1)
```

经过以上处理，所有列的各元素均为数值型，因此可用 astype 函数将其设置为 float 型。此外，我们还可以用 lambda 函数，将 DataFrame 的每一列转换为数值类型[①]：

```
In [82]: data=data.apply(lambda x: pd.to_numeric(x))
```

上述代码，每个 x 实例表示一列，to_numeric 函数将每一类的元素转换为最相近的数据类型（该例为 float）。

---

① 若 DataFrame 存在非数值型元素，转换时会报错。pandas 0.17 及以上版本，可使用 df1 = df.apply(pd.to_numeric, errors='coerce') 语句，或提前处理非数值型元素。——译者注

pandas 教程的最后，我们想演示下如何拼接两个 DataFrame 对象，因为在真实应用场景，可能会用到这项操作。我们随机选取元素，再创建一个小型的 DataFrame：

```
In [83]: data1 = pd.DataFrame(columns=[i for i in xrange(1559)])
         data1.loc[len(data1)] = [random.randint(0,1) for r in xrange(1558)]+[1]
         data1.loc[len(data1)] = [random.randint(0,1) for r in xrange(1558)]+[1]
```

上述代码生成一个包括两行数据的新表格。我们可以用 concat 函数，将其合并到原 DataFrame 中，将 data1 的各行拼接到 data 的下面：

```
In [85]: print len(data)
         datatot = pd.concat([data[:],data1[:]])
         len(datatot)

         2362

Out[85]: 2364
```

执行上述操作后，我们会发现 datatot 比 data 增加了两行（注意 data 的行数与刚开始的不同，因为我们后来删除了它含有 NaN 元素的行）。

1.2.3 matplotlib 教程

matplotlib.pyplot 库，类似于 MATLAB，提供多种将数据绘制成图的方法。由于后续章节的一些数据分析结果要用它实现可视化，因此我们有必要用一个简短的例子，解释后面即将用到的所有 matplotlib 代码：

```
In [1]: import matplotlib.pyplot as plt

In [2]: plt.plot([10,5,2,4],color='green',label='line 1', linewidth=5)
        plt.ylabel('y',fontsize=40)
        plt.xlabel('x',fontsize=40)
        plt.axis([0,3, 0,15])
        plt.show()

In [5]: fig = plt.figure(figsize=(10,10))
        ax = fig.add_subplot(111)
        ax.set_xlabel('x',fontsize=40)
        ax.set_ylabel('y',fontsize=40)
        fig.suptitle('figure',fontsize=40)
        ax.plot([10,5,2,4],color='green',label='line 1', linewidth=5)
        fig.savefig('figure.png')
```

导入该库之后（导入为 plt），初始化 figure 对象（fig），添加 axis 对象（ax）。每条线是通过 ax.plot()命令绘制到 ax 对象之中，每条线称为句柄（handle）。然后，matplotlib.pyplot 记录下面所有指令，并将其绘制到 figure 对象之中。该例中，用 plt.show()命令直接在终端显示绿色折线，并用 fig.savefig()函数将其保存为 figure.png 文件。运行结果见图 1.1。

接下来这个例子讲解如何用一条命令绘制样式不同的多条曲线，我们用到了 NumPy 数组，见图 1.2。

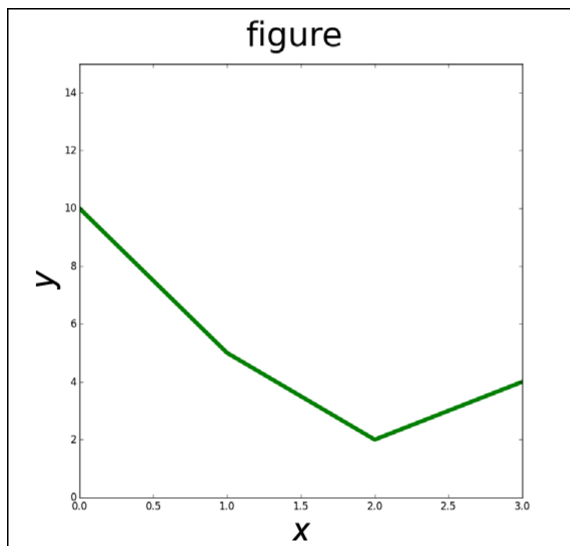


图 1.1　简单图表示例

```
In [8]: import numpy as np
        fig = plt.figure(figsize=(10,10))
        ax = fig.add_subplot(111)
        r = np.arange(0., 10., 0.3)
        p1, = ax.plot(r, r, 'r--',label='line 1', linewidth=10)
        p2, = ax.plot(r, r**0.5, 'bs',label='line 2', linewidth=10)
        p3, = ax.plot(r,np.sin(r),'g^', label='line 3', markersize=10)
        handles, labels = ax.get_legend_handles_labels()
        ax.legend(handles, labels,fontsize=40)
        ax.set_xlabel('x',fontsize=40)
        ax.set_ylabel('y',fontsize=40)
        fig.suptitle('figure 1',fontsize=40)
        fig.savefig('figure_multiplelines.png')
```

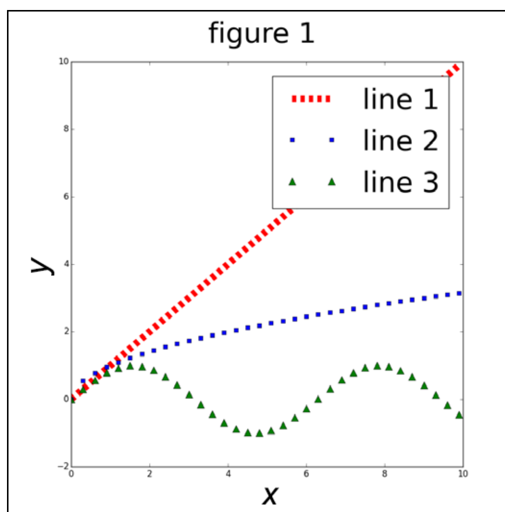


图 1.2　多序列曲线图表示例

注意上述代码中的 get_legend_handles_labels() 函数，返回存储在 ax 对象中的句柄列表和标签，我们需要将这两项返回结果传给 legend 函数完成绘图。符号 “r--” “bs” 和 “g^” 指的是数据点的形状和颜色（分别表示红色矩形、蓝色方形和绿色三角形）。linewidth 参数用来设置线条的密度，markersize 用来设置点的大小。

数据分析结果，另一种常用的可视化方法是散点图，通常用来显示一组数据两个变量的不同取值情况（我们用 NumPy 的 random 子模块生成这样的一组数据）。

```
In [10]: colors = ['b', 'c', 'y', 'm', 'r']
         fig = plt.figure(figsize=(10,10))
         ax = fig.add_subplot(111)
         ax.scatter(np.random.random(10), np.random.random(10), marker='x', color=colors[0])
         p1 = ax.scatter(np.random.random(10), np.random.random(10), marker='x', color=colors[0],s=50)
         p2 = ax.scatter(np.random.random(10), np.random.random(10), marker='o', color=colors[1],s=50)
         p3 = ax.scatter(np.random.random(10), np.random.random(10), marker='o', color=colors[2],s=50)
         ax.legend((p1,p2,p3),('points 1','points 2','points 3'),fontsize=20)
         ax.set_xlabel('x',fontsize=40)
         ax.set_ylabel('y',fontsize=40)
         fig.savefig('figure_scatterplot.png')
```

上述代码，s 选项表示数据点的大小，colors 选项为每组数据点的颜色。我们直接将句柄(p1,p2,p3)传给 legend 函数，见图 1.3。

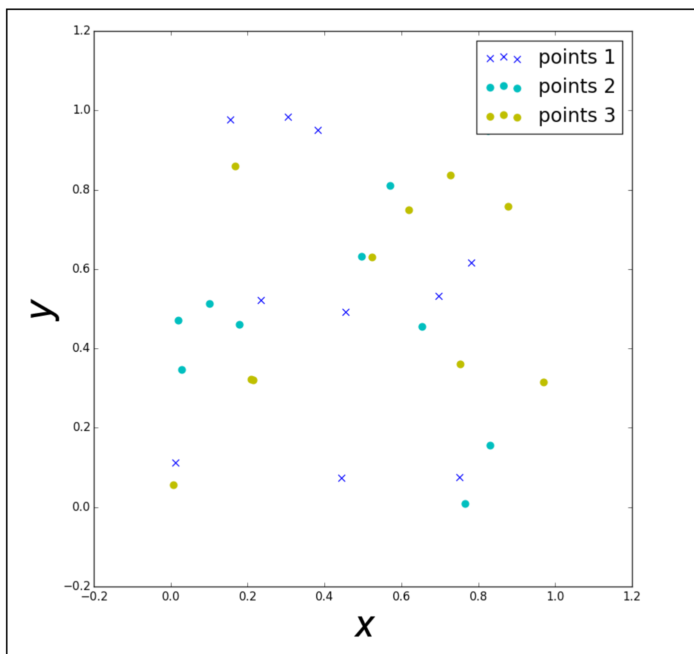


图 1.3　由随机分布的数据点构成的散点图

关于 matplotlib 库的更多细节，我们建议大家读一读网上的相关材料和教程，比如他们官方提供的这份教程：http://matplotlib.org/users/pyplot_tutorial.html。

1.3 本书使用的科学计算库

若想实现本书讲解的机器学习技术，有一些库是必要的。我们简要介绍后续最常用的几个库。

- SciPy 库是基于 NumPy 数组对象开发的一系列数学运算方法。作为开源项目，它得以充分利用世界各地的开发者不断编写出来的新方法。用 SciPy 封装的通用方法开发的 Python 软件，作为高级项目或应用的一部分，足以与 MATLAB、Octave 或 RLab 相媲美。SciPy 提供了多种方法，数据操作、数据可视化和并行计算等方法应有尽有，它使得 Python 语言更加丰富、更具潜力。
- scikit-learn (sklearn)是用 Python 语言实现的开源机器学习模块。它实现了聚类、分类和回归等多种不同的机器学习算法，其中包括**支持向量机**、**朴素贝叶斯**、**决策树**、**随机森林**、***k* 均值**、**基于密度带噪声的空间聚类应用算法**（DBSCAN）。该库和 Python 的 NumPy、SciPy 等数学计算库可通过原生接口对接。虽然该库的大多数方法是用 Python 实现的，但为了提升性能，有些函数是用 **Cython** 实现的。例如，支持向量机和对数几率回归[①]就是用 Cython 编写的，它们对其他几个外部库（LIBSVM、LIBLINEAR）做了封装。
- **自然语言处理工具集**（**NLTK**）包含一系列**自然语言处理**（**NLP**）库和函数。NLTK 的设计初衷是为 NLP 和相关主题的研究、教学提供支持。这些主题有人工智能、认知科学、信息检索、语言学和机器学习。它还有一个特色是提供一系列文本处理函数，可实现分词（tokenization）、提取词干（stemming）、标注（tagging）、句法分析（parsing）、语义推理（semantic reasoning）和分类功能。NLTK 还提供示例代码和数据，可接入 50 多种语料库和词汇数据库。
- Scrapy 是用 Python 实现的开源 Web 爬虫（crawler）框架。它最初是为网站抓取设计的，但是作为通用型爬虫，它也适合用 API 抽取数据。Scrapy 项目意在实现网络蜘蛛（spider）功能，蜘蛛的行为通过一组指令来控制。它的另一特色是提供 Web 抓取 shell，开发人员在编码实现概念之前，可先用 shell 做测试。Scrapy 如今由 Scrapinghub Ltd.公司负责，这是一家提供 Web 抓取技术的开发和服务的公司。
- Django 是用 Python 实现的开源 Web 应用框架，遵从模型-视图-控制器（model-

① logistic regression，大抵有几种译法：逻辑回归、逻辑斯蒂回归、逻辑斯蒂克回归、对数几率回归，还有干脆不译的。这里取对数几率回归。——译者注

view-controller）架构模式。Django 的设计初衷是创建功能复杂、由数据库驱动的网站。开发人员可通过它提供的管理界面管理应用。在管理界面可创建、读取、删除或更新应用所使用的数据。当前，一些知名网站是用 Django 驱动的，比如 Pinterest、Instagram、Mozilla、The Washington Times（《华盛顿邮报》官网）和 Bitbucket。

1.4 机器学习的应用场景

机器学习也并不是魔法，不是所有与数据相关的问题都能受益于它，因此在入门章节的最后，讲清楚机器学习技术的最佳应用场景非常有必要。

- 规则不可能用编码实现：一些需要由人完成的任务（例如，判断邮件是否是垃圾邮件）无法有效地用简单的规则来实现。实际上，多种因素可能影响其解决方案，如果规则依赖于大量因素，人工实现这些规则非常困难。
- 解决方案无法扩展：人工根据特定数据做出决策非常耗时，但机器学习技术却有很好的扩展性。例如，机器学习算法可以高效地遍历百万封邮件，判断它们是不是垃圾邮件。

然而，如果仅用数学规则、计算或预先确定的模式就能做到准确预测，并且实现这些方法无须用机器驱动的学习技术，那么你就没必要使用高级机器学习技术（你也不应该使用）。

1.5 小结

在本章中，我们介绍了基本的机器学习概念和术语，这些知识是后续章节的基础。我们还介绍了机器学习专业人士准备数据、处理数据和实现数据可视化最常用的几个库（NumPy、pandas 和 matplotlib）。此外，后面要用到的其他几个 Python 库，我们也一并做了简要介绍。

学完这一章，你应该大体了解了机器学习技术的实际用途。你应该熟悉了常用的数据处理方法，能够将数据转换为机器学习算法可以处理的格式。下一章，我们来向大家解释主要的无监督学习算法以及如何用 sklearn 库实现它们。

第 2 章
无监督机器学习

第 1 章已介绍过，无监督学习的目的是从未标记数据发现富有洞察力的信息。大型数据集（指数据点和特征的数量都很多）往往缺乏内在结构（我们称其为“非结构化”），乍一眼看上去很难看出任何信息。遇到这种情况，就要用无监督机器学习技术，突显隐藏在数据中的内在结构（聚类），或在不丢失相关信息的前提下降低数据的复杂度（降维）。本章重点介绍主要聚类算法（第 1 部分）和降维方法（第 2 部分）。它们的不同点和各自的优点，我们会用实际的例子加以说明。实现这些例子要用到 Python 的几个科学计算库。所有代码均可从我的 GitHub 主页下载，地址是 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_2/。现在我们开始讲聚类算法。

2.1 聚类算法

聚类算法（clustering algorithm），将数据重组为按某种方式排列的多个子集（簇，cluster），以便从数据中推断出有意义的结构。我们可以将簇定义为一组具有某些相似特征的数据点。量化数据点之间相似度的方法，决定着聚类算法的种类。

根据处理数据时所使用的度量方法或做出的假设，我们可将聚类算法分成不同的种类。接下来讨论时下最常用的几大方法：分布方法、质心点方法、密度方法和层次方法（hierarchical methods）。每种聚类方法，我们都会详细讲解它的一种算法实现。我们首先讨论分布方法。后面我们将通过实际的例子比较不同算法的性能。本章代码的 IPython notebook 版本和纯 Python 代码版本，都已放到我的 GitHub 主页该书的文件夹下，详见 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_2/。

2.1.1　分布方法

这类方法假定数据来自某种分布，并用最大期望算法寻找分布参数的最优参数值。下面我们讨论最大期望算法和**高斯混合聚类**（Gaussian clustering）。

1．最大期望算法

最大期望算法（expectation maximization），是用于寻找参数分布模型的最大似然估计，分布模型依赖于隐变量[①]（变量的值无法观测到）。最大期望算法的每次迭代包括两个步骤：**期望步**（E-step），用参数的当前值构造对数似然函数（log-likelihood function）；**最大化步**（M-step），计算得到使 E 步对数似然度最大的新参数值。

给定一个由 N 个元素组成的数据集 $\left\{x^{(i)}\right\}$ $i = 1,\cdots,N$，其对数似然度为：

$$l(\theta)=\sum_{i=1}^{N}\log p\left(x^{(i)};\theta\right)=\sum_{i=1}^{N}\log\sum_{z}p\left(x^{(i)},z^{(i)};\theta\right)$$

其中，θ 为分布的参数，$z^{(i)}$ 表示所谓的隐变量。

我们希望在不知道 $z^{(i)}$（无法观测的变量）取值的情况下，找到最大化对数似然度的参数值。假定我们有一种 $z^{(i)}$ 上的分布，和 $z^{(i)}$ 的概率密度函数 $Q(z^{(i)})$，$\sum_{i=1}Q\left(z^{(i)}\right)=1$。因而：

$$Q\left(z^{(i)}\right)=\frac{p\left(x^{(i)},z^{(i)};\theta\right)}{p\left(x^{(i)};\theta\right)}=p\left(z^{(i)}\middle|x^{(i)};\theta\right)$$

上式中的 $Q(z^{(i)})$为隐变量 $z^{(i)}$ 在 $x^{(i)}$ 发生这一前提下的后验概率，以 θ 为参数。最大期望算法用到了 Jensen 不等式，它确保执行以下两步：

$$1.Q\left(z^{(i)}\right)=p\left(z^{(i)}\middle|z^{(i)};\theta\right)$$

$$2.\sum_{i}\sum_{z^{(i)}}Q\left(z^{(i)}\right)\log\frac{p\left(x^{(i)},z^{(i)};\theta\right)}{Q\left(z^{(i)}\right)}$$

对数似然函数收敛，得到最大对数似然度，这时对应的 θ 就是我们想求的。

2．高斯混合聚类算法

高斯混合聚类算法（Mixture of Gaussians），用混合的多个高斯分布来模拟整个数据集。

① 隐变量在英文中称为 hidden variable 或 latent variable。——译者注

因而，簇的数量由模型所包含的高斯分布的数量来决定。给定由 N 个元素组成的数据集 $\left\{x^{(i)}\right\}$，$i=1,\cdots,N$，其中每个 $x^{(i)}\in R^d$ 为高斯混合模型中一个由 d 个特征组成的向量，并且：

$$p\left(x^{(i)}\middle|\bar{\mu},\bar{\Sigma}\right)=\sum_{k=1}^{K}p\left(x^{(i)}\bar{\mu},\bar{\Sigma},z^{(i)}=k\right)p\left(z^{(i)}=k,\phi\right)=\sum_{k=1}^{K}p\left(x^{(i)}\middle|\mu_k,\Sigma_k\right)\phi_k$$

其中：

- $z^{(i)}\in 1,\cdots,K$ 为隐变量，表示每个 $x^{(i)}$ 是由哪个高斯成分（Gaussian component）生成的 μ={ $\mu_1,\cdots,\mu_K$ }为各高斯成分的均值参数。
- Σ={ $\Sigma_1,\cdots,\Sigma_K$ }为各高斯成分的方差参数。
- ϕ_k 为混合权重[①]，表示随机选取的 $x^{(i)}$由高斯成分 k 生成的概率，其中 $\sum_{k=1}^{K}\phi_k=1$，ϕ={ $\phi_1,\cdots,\phi_K$ }为各权重的集合。
- $p\left(x^{(i)}\middle|\mu_k,\Sigma_k\right)=\frac{1}{(2\pi)^{d/2}\,|\Sigma_k^{1/2}}e^{-\frac{1}{2}\left(x^{(i)}-\mu_k\right)^T\Sigma_k^{-1}\left(x^{(i)}-\mu_k\right)}$ 表示每个数据点 $x^{(i)}$ 对应的以(μ_k , Σ_k)为参数的高斯成分 k。

高斯混合模型的参数有 ϕ、μ 和 Σ。为了估计这些参数，我们可以将数据集的对数似然函数写为：

$$l\left(\phi,\bar{\mu},\bar{\Sigma}\right)=\sum_{i=1}^{N}\log\left(p\left(x^{(i)}\middle|\bar{\mu},\bar{\Sigma}\right)\right)=\sum_{i=1}^{N}\log\sum_{i=1}^{N}p\left(x^{(i)}\bar{\mu},\bar{\Sigma},z^{(i)}=k\right)p\left(z^{(i)}=k,\phi\right)$$

我们采用前一节讲的最大期望算法寻找参数值，参数的对应关系为 θ= (μ, Σ)、$Q(z^{(i)})=p(z^{(i)},\phi)$。

参数的初始值我们不知道，只好先用猜测的值，然后迭代以下步骤直到收敛。

（1）**期望步**：根据贝叶斯定理计算 $z^{(i)}$ 的后验概率，更新权重 $W_k^{(i)}=p\left(z^{(i)}=k\middle|x^{(i)},\phi,\bar{\mu},\bar{\Sigma}\right)$：后验概率计算方法如下：

$$p\left(z^{(i)}=k\middle|x^{(i)},\phi,\bar{\mu},\bar{\Sigma}\right)=\frac{p\left(x^{(i)}\middle|\mu_k,\Sigma_k\right)\phi_k}{\sum_{l=1}^{K}p\left(x^{(i)}\middle|u_l,\Sigma_l\right)\phi_l}$$

（2）**最大化步**：将参数更新为如下形式（下面 3 个式子是通过求最大值得到的，具体

① 混合权重，原文为“mixture weight”。周志华老师的《机器学习》207 页将其称作“mixture coefficient”，译为“混合系数”。——译者注

方法是令似然函数的导数为 0，分别对 μ、Σ 和ϕ求偏导得到的）：

$$\mu_k = \frac{\sum_{i=1}^{N} w_k^{(i)} x^{(i)}}{\sum_{i=1}^{N} w_k^{(i)}}$$

$$\Sigma_k = \frac{\sum_{i=1}^{N} w_k^{(i)} \left(x^{(i)} - \mu_k\right)\left(x^{(i)} - \mu_k\right)^T}{\sum_{i=1}^{N} w_k^{(i)}}$$

$$\phi_k = \frac{\sum_{i=1}^{N} w_k^{(i)}}{N}$$

注意，由于隐变量 $z^{(i)}$ 未知，因此需要使用最大期望算法。否则，这个问题就变为有监督学习问题，$z^{(i)}$ 则表示训练集每个数据点的标签（并且使用的有监督算法将会是高斯判别分析[①]）。因而，高斯混合聚类是无监督算法，其目标是找到 $z^{(i)}$，也就是每个数据点 $x^{(i)}$ 对应的 K 个高斯成分。事实上，通过计算 K 个类别每个类别的后验概率 $p\left(z^{(i)} = k \middle| x^{(i)}, \phi, \overline{\mu}, \overline{\Sigma}\right)$，我们可以将每个 $x^{(i)}$ 划分到后验概率最高的类别 k 中。在一些应用场景中，该算法能够胜任数据聚类（或标记数据）任务。

我们举个实际的例子，来说明高斯混合聚类算法可能的应用场景有哪些。比如，教授拿到了两个班的成绩，但是没有标出学生的班级。假定每个班的成绩都服从高斯分布，他想将成绩按班级分开，这时就可以用高斯混合聚类。再举一个例子，我们从两个国家采集了一组身高数据，假定每个国家人口的身高都服从高斯分布，如何根据每个人的身高数据判断他们来自哪个国家，这个问题可用该算法来解决。

2.1.2 质心点方法

质心点方法（centroid methods）用到以下技术：寻找各簇的中心，将数据点划分到离它最近的簇，最小化簇的质心点和被划归到该簇的数据点之间的距离。这是一个最优化问题，最后得到的质心点为向量，它们也许不是原数据集的数据点。该类聚类方法，簇的数量是待估参数，我们需要为其指定一个初始值。该类方法最终生成的各个簇，它们的大小通常差不多，这样也就没必要精确界定各簇之间的边界。这个最优化问题可能会得到局部最优解，这表明初始值不同，得到的簇会有轻微的不同。最常用的质心点聚类方法为 k **均值算法**（Lloyd 算法），其距离度量方法

① 英文为“Gaussian discriminant analysis”。——译者注

为**欧几里得范数**（Euclidean norm），该算法要最小化该种距离。其他寻找质心点的方法，用各簇的中值（**k 中值聚类**）或强制使用实际存在的数据点作为质心点。进一步讲，这些方法还有一些变种，它们的区别在于最初质心点的定义方式（**k 均值++**[①]或**模糊 c 均值**）不同。

k 均值算法

该算法尝试根据每个簇数据点之间的平均距离寻找质心点，使被划归到各簇的数据点到质心点之间的距离最小。我们可将其与解决分类问题的 k 近邻算法对照来看，两者之间存在联系。聚类的结果可以表示为**维诺图**。（Voronoi diagram，一种根据距离一组点的远近，比如我们这里各簇的质心点，将空间划分为不同区域的方法）。给定数据集$\{x^{(i)}\}, i\in 1,\cdots,N$，该算法要求预先选择一组质心点 K。我们将每个簇各数据点到质心点（距离）的均值表示为 μ_j，$j\in 1,\cdots,K$，为 μ_j 赋予随机值。然后，迭代以下步骤直到算法收敛。

（1）对于每一个数据点 i，计算 i 跟每个质心点 j 之间的欧氏距离，寻找质心点索引 d_i，使得每个数据点和质心点之间的距离最小化：$|\mu_j - x^{(i)}|$，$j\in 1,\cdots,K$。

（2）对于每一个质心点 j，重新计算 d_ij 等于 j 的这些数据点（数据点属于均值为 μ_j 的簇）到质心点距离的均值：

$$\mu_j = \frac{\sum_{i=1}^{N} \delta_{d,j} x(i)}{\sum_{i=1}^{N} \delta_{d_ij}}$$

易知该算法收敛于以下函数：

$$F = \sum_{i=1}^{N} \left| x^{(i)} - \mu_{d_i} \right|$$

随着迭代次数的增加，函数值单调递减。既然 F 为非凸函数，无法保证最终得到的最小值为全局最小值。为了避免聚类结果为局部最小值，我们通常随机选取不同的初始均值，多运行几次算法。然后，从中选取使得 F 函数值较小的那一种作为解决方案。

2.1.3 密度方法

密度方法（density methods）所依据的思想是，数据点稀疏的区域通常被视为不同簇之间的边界（或噪声），而簇的中心数据点则通常出现在密度高的区域。常用的密度方法叫作**基于密度带噪声的空间聚类应用算法**（**DBSCAN**），它用特定的距离阈值定义两个数据点之

---

① 英文为“k-means++”。——译者注

间的可连接性（因此，该方法类似于层次算法；见第 3 章）。只有满足一定的紧密程度，两个数据点才被看作是相互连接的（属于同一个簇）——在一定半径范围之内，近邻的数量必须高于某个阈值。另一种常用方法是均值漂移（mean-shift），它将每个数据点划归到在近邻当中密度最高的簇。由于用核密度估计方法计算密度，时间开销较大，均值漂移通常比 DBSCAN 或质心点方法速度慢。密度聚类方法的主要优点在于，方法本身能够定义任意形状的簇，能够决定将数据集分成几个簇最合适，而不必将簇的数量作为参数预先设置好，因而它适合处理簇的形状和数量都未知的数据集。

均值漂移

均值漂移是一种非参数估计算法，它寻找数据集核密度函数的局部最大值。所找到的局部最大值可以看作是$\{x^{(i)}\}$，$i=1,\cdots,N$ 各簇的中心，局部最大值的数量即为簇的数量。若要用均值漂移实现聚类，每个数据点 $x^{(l)}\in R^d$ 需要与其近邻的密度建立起关系：

$$f\left(x^{(l)}\right)=\frac{1}{Nh^d}\sum_{i=1}^{N}K\left(\frac{x^{(l)}-x^{(i)}}{h}\right)$$

其中，h 表示带宽（bandwith）；它估计的是近邻的半径，在半径范围内的点影响密度值 $f(x^{(l)})$（也就是说，其他数据点对 $f(x^{(l)})$ 有微弱影响，可以忽略）。K 为满足以下条件的核函数：

- $\int_{R\wedge d}K\left(x^{(i)}\right)=1$
- $K\left(x^{(i)}\right)\geqslant 0,i\in 1,\cdots,N$

核函数 $K(x^{(i)})$ 最常见的形式有：

- $K\left(x^{(i)}\right)=e^{\frac{-\left(x^{(i)}\right)^2}{2\sigma^2}}$：高斯内核
- $K\left(x^{(i)}\right)=\begin{cases}\frac{3}{4}\left(1-\left(x^{(i)}\right)^2\right) if \left|x^{(i)}\right|\leqslant 1\\ 0 \quad else\end{cases}$：Epanechnikov 内核

均值漂移算法要求 $f(x^{(l)})$ 取得最大值，这可以转换为如下等式（还记得吧，在函数分析里，函数最大值在导数为 0 时取到）：

$$\nabla f\left(x^{(l)}\right)=0\to x^{(l)}=\frac{\sum_{i=1}^{N}K'\left(\frac{x^{(l)}-x^{(i)}}{h}\right)x^{(i)}}{K'\left(\frac{x^{(l)}-x^{(i)}}{h}\right)}$$

其中，K'为核密度函数 K 的导数。

因而，对下面这个等式进行迭代，就可以找到特征向量 $x^{(i)}$ 对应的局部最大位置。

$$x_{t+1}^{(l)} = x_t^{(l)} + \frac{\sum_{i=1}^{N} K'\left(\frac{x_t^{(l)} - x^{(i)}}{h}\right) x^{(i)}}{K'\left(\frac{x_t^{(l)} - x^{(i)}}{h}\right)} - x_t^{(l)} = x_t^{(l)} + m\left(x_t^{(l)}\right)$$

其中，$m(x_t^{(l)})$叫作均值漂移向量。当 $t=a$ 时，条件 $\nabla f\left(x_a^{(l)}\right) = 0 \rightarrow m\left(x_a^{(l)}\right)=0$ 满足，算法收敛。

有上面这个公式做基础，我们现在可以借助图 2.1 解释该算法。第 0 次迭代时，$t=0$，原始数据点 $x^{(l)}, l \in 1, \cdots, N$（红色）散布于数据空间。计算均值漂移向量 $m\left(x_0^{(l)}\right) = m\left(x^{(l)}\right)$，$l \in 1, \cdots, N$，为移动后的数据点打上差号标记，以跟踪它们在算法迭代过程位置上的变动。第 1 次迭代，$t=1$，用前面提到的公式得到的数据集作为这一次迭代所使用的数据集，计算各数据点的位置，图中用带有+号的圈圈表示经过这次迭代后数据点所处位置。

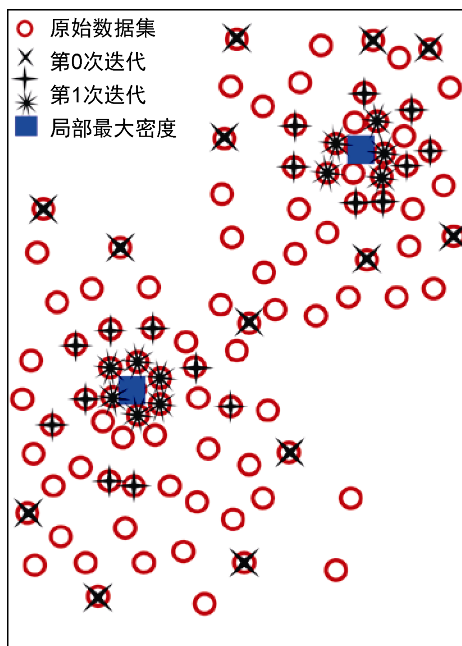


图 2.1 均值漂移算法迭代过程，数据点的变动

在图 2.1 中，第 0 次迭代，原始数据集用带有红色①差号的圆圈表示，第 2、K 次迭代，数据点（分别带有+和*符号）向由蓝色方块标识的局部最大密度移动。

第 K 次迭代，计算得到新数据点 $x_k^{(l)}$，$l \in 1, \cdots, N$，上图用带*的圆圈表示。$x_k^{(l)}$ 所对应的密度函数 $f\left(x_k^{(l)}\right)$ 的值大于上一次迭代的函数值，因为该算法的目的是最大化函数值。经过 K 次迭代后，原始数据集对应②数据点 $x_k^{(l)}$ $l \in 1, \cdots, N$，这些数据点收敛到图 2-1 用蓝色方块标记的位置。特征向量 $x^{(i)}, l \in 1, \cdots, N$ 向两个不同的局部最大值塌陷，这两个极值表示两个簇。

合理使用该方法，需注意以下事项。

均值漂移算法唯一要求的参数是带宽 h，巧妙地调试该参数，才能得到理想的分簇结

① 彩色图片见本书配套 PDF 文件。下同。——译者注

② 原文为“clearly associated with”。——译者注

果。从实际情况来看，h 太小，得到的簇数量较多，而 h 太大，也许会将多个不同的簇合并到一起。另外，还要注意，如果特征向量的维度 d 很大，均值漂移算法也许得到的结果较差，因为空间维度高，相应地局部最大值的数量也很多，迭代函数可能收敛得过快。

2.1.4 层次方法

层次方法也叫作基于连接性（connectivity-based）的聚类，它根据某种距离度量方法，选取具有满足相似标准的元素形成簇：距离较近的元素聚集在同一个簇，而距离较远的元素分到不同的簇。这类算法又可分为两种：**分割聚类**（divisive clustering）和**合成聚类**（agglomerative clustering）。分割聚类，一开始将整个数据集分到一个簇，接着将其分到两个不那么相似（距离稍远）的簇。分割过程，得到的每一部分再次分割，直到每个数据点自成一簇。合成聚类是最常用的聚类方法，它从小处着眼，首先将每个数据点各分到一个簇。然后，根据相似度，将这些簇合并，直到所有的数据点都被分到同一个簇。这两种方法通过迭代形成了位于不同层次的簇，因此也称为**层次聚类**，参见图 2.2。顺便提一下，这种表示层次的方法叫作**树状图**（dendrogram）。横轴表示数据集的数据点，纵轴表示距离。每条水平线段表示一个簇，纵轴表示哪些元素或簇因为相似而合并为另一个层次较高的簇。

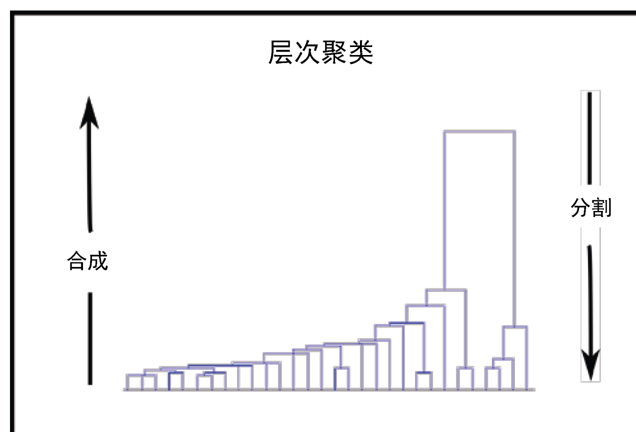


图 2.2

图 2.2，合成聚类方法，一开始簇的数量跟数据点的数量一样多，最终得到的是一个包罗所有数据点的簇。反之，分割聚类方法，一开始只有一个簇，结束时得到的每个簇只包含一个数据点。

我们需要使用特定标准来终止合成/分割策略，从而得到最终的分簇结果。我们用**距离标准**来指定两个簇无法合成一个簇的最大距离，用**簇数量标准**来指定停止层次聚类算法继续合并或分割簇的最大簇数量。

合成方法的一种算法描述如下。

（1）将数据集$\{x^{(i)}\}$，$i\in 1,\cdots,N$ 的每个元素 i 划分到不同的簇中，使得各个元素自成一簇。

（2）计算所有簇两两之间的距离，将距离最近的两个簇合并为一个簇，簇的总数减 1。

（3）计算新得到的簇和其他簇之间的距离。

（4）重复步骤（2）和（3），直到得到一个包含所有 N 个元素的簇。

簇 $C1$ 和 $C2$ 之间的距离 $d(C1,C2)$，实际计算时，用两个数据点 $c1\in C1$，$c2\in C2$ 之间的距离来代替。对于包含多个数据点的簇，有必要根据某种标准选择数据点来计算距离。$C1$ 和 $C2$ 两个簇之间常用的链接标准如下。

- **单链接**（single linkage）：$C1$ 中的任意元素和 $C2$ 中的任意元素之间的最小距离表示为：

$$d(C1,C2)=\min\left\{d\left(c1,c2\right):c1\in C1,c2\in C2\right\}$$

- **全链接**（complete linkage）：$C1$ 中的任意元素和 $C2$ 中的任意元素之间的最大距离表示为：

$$d\left(C1,C2\right)=\max\left\{d\left(c1,c2\right):c1\in C1,c2\in C2\right\}$$

除权配对法（UPGMA）或**均链接**（average linkage）：$C1$ 中的任意元素和 $C2$ 中的任意元素之间的平均距离表示为 $d\left(C1,C2\right)=\frac{1}{|N_{c1}||N_{c2}|}\sum_{c1\in C1}\sum_{c2\in C2}d\left(c1,c2\right)$，其中，$|N_{c1}|$、$|N_{c2}|$分别为 $C1$、$C2$ 中元素的数量。

- **Ward 算法**：它将不会增加异质性（heterogeneity）的簇合并。它的目标是，在最小化增加 variation measure 的前提下，合并 $C1$ 和 $C2$ 两个簇。variation measure 叫作合并代价，用$\Delta(C1,C2)$表示。对于该算法，距离替换为合并代价，其公式如下：

$$\Delta\left(C1,C2\right)=\frac{N_{c1}N_{c2}}{N_{c1}+N_{c2}}\left|\mu_{c1}-\mu_{c2}\right|^2 \mu_{c1}=\frac{1}{N_{c1}}\sum_{c1\in C1}c1,\mu_{c2}=\frac{1}{N_{c2}}\sum_{c2\in C2}c2$$

其中，$|N_{c1}|$、$|N_{c2}|$分别为 $C1$、$C2$ 中元素的数量。

实现层次聚类算法，$d(c1,c2)$有不同的度量方法，最常用的是欧氏距离：

$$d\left(c1,c2\right)=\sqrt{\sum_i\left(c1_i-c2_i\right)^2}$$

请注意层次聚类方法从时间方面来讲不是非常高效，因此不适合大型数据集聚类。并且，为存在错误的数据点（异常值）聚类，鲁棒性不好，它也许会误把几个簇合并。

聚类方法的训练和对比

我们生成一个数据集，比较前面介绍的几种聚类方法。我们从均值为 $\mu_1 = [10,0]$和 $\mu_2 = [0,10]$，方差为$\sigma_1 = \sigma_2 = \begin{bmatrix} 3 & 1 \\ 1 & 4 \end{bmatrix}$的两个二维正态分布，随机选取数据点形成数据集。

我们用 NumPy 库生成数据点，用 matplotlib 库绘制图像。

```
from matplotlib import pyplot as plt
import numpy as np
np.random.seed(4711) # for repeatability
c1 = np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]],
size=[100,])
l1 = np.zeros(100)
l2 = np.ones(100)
c2 = np.random.multivariate_normal([0, 10], [[3, 1], [1, 4]],
size=[100,])
#add noise:
np.random.seed(1) # for repeatability
noise1x = np.random.normal(0,2,100)
noise1y = np.random.normal(0,8,100)
noise2 = np.random.normal(0,8,100)
c1[:,0] += noise1x
c1[:,1] += noise1y
c2[:,1] += noise2

fig = plt.figure(figsize=(20,15))
ax = fig.add_subplot(111)
ax.set_xlabel('x',fontsize=30)
ax.set_ylabel('y',fontsize=30)
fig.suptitle('classes',fontsize=30)
labels = np.concatenate((l1,l2),)
X = np.concatenate((c1, c2),)
pp1= ax.scatter(c1[:,0], c1[:,1],cmap='prism',s=50,color='r')
pp2= ax.scatter(c2[:,0], c2[:,1],cmap='prism',s=50,color='g')
ax.legend((pp1,pp2),('class 1', 'class2'),fontsize=35)
fig.savefig('classes.png')
```

我们为两个类别分别添加了服从正态分布的噪音，使例子看起来更加真实。结果如图 2.3 所示。

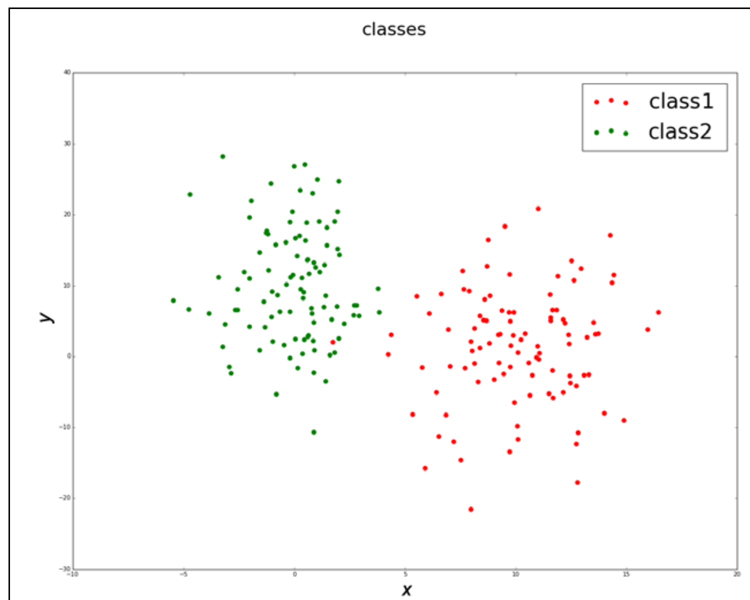


图 2.3 含噪声、服从正态分布的两个类别

我们用 sklearn 和 scipy 库实现几种聚类方法，绘图仍用 matplotlib 库：

```
import numpy as np
from sklearn import mixture
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import fcluster
from sklearn.cluster import KMeans
from sklearn.cluster import MeanShift
from matplotlib import pyplot as plt

fig.clf()#reset plt
fig, ((axis1, axis2), (axis3, axis4)) = plt.subplots(2, 2, sharex='col',
sharey='row')

#k-means
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
pred_kmeans = kmeans.labels_
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='prism') # plot
points with cluster dependent colors
axis1.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='prism')
axis1.set_ylabel('y',fontsize=40)
```

```
axis1.set_title('k-means',fontsize=20)

#mean-shift
ms = MeanShift(bandwidth=7)
ms.fit(X)
pred_ms = ms.labels_
axis2.scatter(X[:,0], X[:,1], c=pred_ms, cmap='prism')
axis2.set_title('mean-shift',fontsize=20)

#gaussian mixture
g = mixture.GMM(n_components=2)
g.fit(X)
pred_gmm = g.predict(X)
axis3.scatter(X[:,0], X[:,1], c=pred_gmm, cmap='prism')
axis3.set_xlabel('x',fontsize=40)
axis3.set_ylabel('y',fontsize=40)
axis3.set_title('gaussian mixture',fontsize=20)

#hierarchical
# generate the linkage matrix
Z = linkage(X, 'ward')
max_d = 110
pred_h = fcluster(Z, max_d, criterion='distance')
axis4.scatter(X[:,0], X[:,1], c=pred_h, cmap='prism')
axis4.set_xlabel('x',fontsize=40)
axis4.set_title('hierarchical ward',fontsize=20)
fig.set_size_inches(18.5,10.5)
fig.savefig('comp_clustering.png', dpi=100)
```

上述代码中，我们需要为 k 均值函数和高斯混合模型为指定簇的数量（n_clusters=2、n_components=2），将均值漂移算法的带宽设置为 7（bandwidth=7）。层次聚类算法使用 Ward 链接来定义距离，终止层次算法的最大（Ward 链接）距离 max_d，我们设置为 110。fcluster 函数预测每个数据点属于哪个簇。k 均值和均值漂移方法的聚类结果用 labels_属性就能获取到，而高斯混合模型则需要使用 predict 函数。k 均值、均值漂移和高斯混合方法用 fit 函数训练，而层次聚类算法用 linkage 函数训练。上述代码输出图 2.4。

在图 2.4 中，均值漂移和层次聚类方法将数据分成两个簇，因此我们选取的参数值（带宽和最大距离）合适。需要注意的是，层次聚类方法，我们是根据下列代码生成的树状图，来选择最大距离这一参数的取值：

我们通过 truncate_mode='lastp'标记可指定将最后多少次合并绘制成图（该例中 p=12）。图 2.4 清楚地表明，当距离在 100 到 135 之间时，只剩下两个簇。

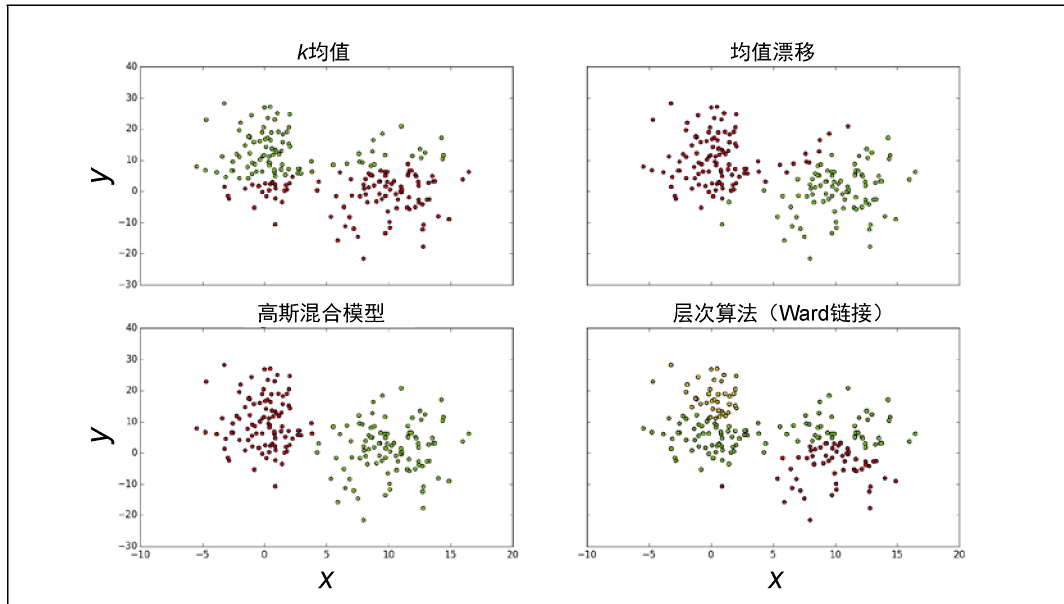


图 2.4　用 *k* 均值、均值漂移、高斯混合模型和层次算法（Ward 链接）对两类数据聚类结果

```
from scipy.cluster.hierarchy import dendrogram
fig = plt.figure(figsize=(20,15))
plt.title('Hierarchical Clustering Dendrogram',fontsize=30)
plt.xlabel('data point index (or cluster index)',fontsize=30)
plt.ylabel('distance (ward)',fontsize=30)
dendrogram(
    Z,
    truncate_mode='lastp', # show only the last p merged clusters
    p=12,
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,
)
fig.savefig('dendrogram.png')
```

在图 2.5 中，横轴坐标标签（小括号中的数字），表示最后 12 次合并之前，每个簇所包含的数据点数量。

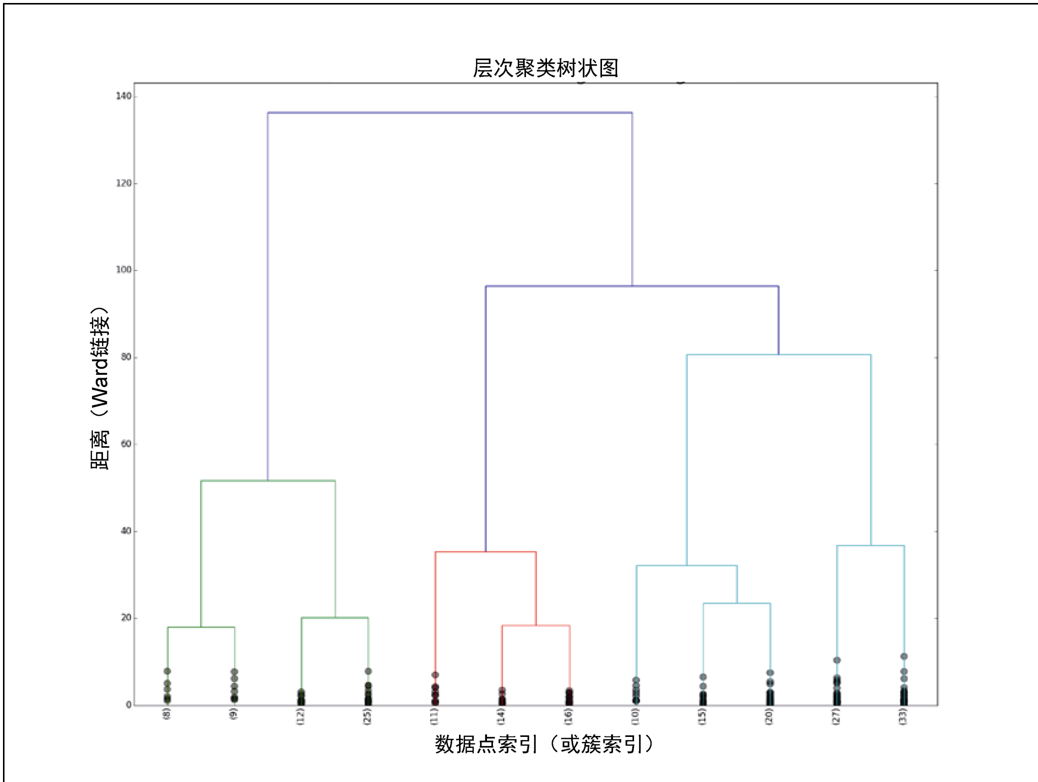


图 2.5　最后 12 次合并的层次聚类树状图

除高斯混合模型之外，其他 3 种算法将一些数据点分错簇，这一点 k 均值和层次聚类方法表现最为突出。上述结果证实，高斯混合模型是鲁棒性最好的方法，因为数据集所属的分布跟假定的相同。为了评估聚类方法的效果，scikit-learn 提供了几种量化分割（partition）正确性的方法：v-measure、完整性和同质性。这些评估方法要用到每个数据点的实际类别，因此它们也被称为外部验证机制，它们需要用到聚类时用不到的额外信息。同质性 h（homogeneity）取值介于 0～1，它衡量的是每个簇是否只包含同一类别的数据点。完整性 c 的取值同样是介于 0～1，它衡量的是同一类别的所有数据点是否划分到同一个簇之中。假定聚类后，将每个数据点分到不同的簇，每个簇只包含一个类别，同质性就为 1，但是除非每个类别只包含一个数据点，否则完整性就非常低，因为误将同一类别的数据点分散到多个簇之中去了。反之，如果聚类结果将分属于多个类别的数据点划分到同一簇，当然其完整性为 1，但同质性极低。这两个值的计算公式类似，如下所示：

$$h=1-\frac{H\left(C_l|C\right)}{H\left(C_l\right)},c=1-\frac{H\left(C|C_l\right)}{H\left(C\right)}$$

其中：

- 给定聚类任务，$H(C_l|C)$为各类别 C_l 的条件熵：$H\left(C_l|C\right)=-\sum_{p=1}^{|C_l|}\sum_{c=1}^{|C|}\frac{N_{pc}}{N}\log\left(\frac{N_{pc}}{N_c}\right)$
- 给定类别的所属关系，$H(C|C_l)$为各簇的条件熵：$H\left(C|C_l\right)=-\sum_{p=1}^{|C_l|}\sum_{c=1}^{|C|}\frac{N_{pc}}{N}\log\left(\frac{N_{pc}}{N_p}\right)$
- $H(C_l)$为各类别的熵：$H\left(C_l\right)=-\sum_{p=1}^{|C_l|}\frac{N_p}{N}\log\left(\frac{N_p}{N}\right)$
- $H(C)$为各簇的熵：$H\left(C\right)=-\sum^{|\,|}\frac{N_c}{N}\log\left(\frac{N_c}{N}\right)$
- N_{pc} 指簇 c 中类别为 p 的数据点数量，N_p 指类别为 p 的数据点数量，N_c 指簇 c 数据点数量。

v-measure 为同质性和完整性的调和平均数：

$$v=2\frac{hc}{h+c}$$

这些评估聚类质量的方法，需要使用数据点真正的标签，但实际情况往往做不到。另一种评估方法叫作**轮廓系数**（silhouette cofficient），它仅使用聚类算法用到的数据。它计算的是每个数据点与同簇数据点和其他簇数据点之间的相似度。如果从平均相似度来看，每个数据点与同簇数据点比较起来，比其他簇的数据点更相似，那么各簇的划分就很理想，这时，轮廓系数接近于 1（反之，接近于-1）。给定每个数据点 i 和下面两个量，我们来定义它的公式：

- $d_s(i)$为每个数据点 i 与同簇各数据点之间的平均距离；
- $d_{rest}(i)$为数据点 i 跟其他各簇数据点之间的最小距离。

那么，轮廓系数可以定义为

$s(i)=\frac{d_{rest}(i)-d_s(i)}{max\left(d_s(i),d_{rest}(i)\right)}$，总轮廓系数为所有数据点轮廓系数 $s(i)$的均值。

下面，我们用 sklearn（scikit-learn）计算 4 种聚类算法的 4 个评价指标：

```
from sklearn.metrics import homogeneity_completeness_v_measure
from sklearn.metrics import silhouette_score
res = homogeneity_completeness_v_measure(labels,pred_kmeans)
```

```
print 'kmeans measures, homogeneity:',res[0],' completeness:',res[1],'
v-measure:',res[2],' silhouette score:',silhouette_score(X,pred_kmeans)
res = homogeneity_completeness_v_measure(labels,pred_ms)
print 'mean-shift measures, homogeneity:',res[0],'
completeness:',res[1],' v-measure:',res[2],' silhouette
score:',silhouette_score(X,pred_ms)
res = homogeneity_completeness_v_measure(labels,pred_gmm)
print 'gaussian mixture model measures, homogeneity:',res[0],'
completeness:',res[1],' v-measure:',res[2],' silhouette
score:',silhouette_score(X,pred_gmm)
res = homogeneity_completeness_v_measure(labels,pred_h)
print 'hierarchical (ward) measures, homogeneity:',res[0],'
completeness:',res[1],' v-measure:',res[2],' silhouette
score:',silhouette_score(X,pred_h)
The preceding code produces the following output:
kmeans measures, homogeneity: 0.25910415428 completeness: 0.259403626429
v-measure: 0.259253803872 silhouette score: 0.409469791511
mean-shift measures, homogeneity: 0.657373750073 completeness:
0.662158204648 v-measure: 0.65975730345 silhouette score: 0.40117810244
gaussian mixture model measures, homogeneity: 0.959531296098
completeness: 0.959600517797 v-measure: 0.959565905699 silhouette
score: 0.380255218681
hierarchical (ward) measures, homogeneity: 0.302367273976 completeness:
0.359334499592 v-measure: 0.32839867574 silhouette score:
0.356446705251
```

跟我们前面对 4 类算法聚类图分析一致，高斯混合模型同质性、完整性和 v-measure 值最高（接近于 1）；均值漂移算法各项评价指标还算过得去（约为 0.5）；而 *k* 均值和层次方法较差（约为 0.3）。相反，这几种方法的总轮廓系数都很理想（0.35～0.41），意思是最终得到的簇，定义是比较合理的。

2.2 降维

降维也称作特征抽取，是指将高维数据空间转换为低维的数据空间的操作。降维得到的子空间应该仅包含与原始数据最相关的信息，降维技术分为线性和非线性两大类。降维操作采用多种技术，从大型数据集抽取最相关信息，降低复杂度，同时保留最相关的信息。

最著名的降维算法**主成分分析**（PCA）将原始数据以线性形式映射到维度互不相关的子空间。我们接来下就介绍这种降维算法。这一节的代码 IPython 版和纯 Python 版已放到作者 GitHub 主页图书文件夹：https://github.com/ai2010/machine_learning_for_the_web/tree/

master/chapter_2/。

主成分分析（PCA）

主成分分析算法旨在识别数据集相关信息所在的子空间。实际上，由于某些数据维度的数据点存在相关性，因此 PCA 寻找的是没有相关性、数据不同的维度，这样的维度很少。例如，汽车行驶轨迹可以用一系列变量来描述，比如用 km/h 或 m/s 表示的速度、用经纬度表示的位置、用离开给定点多少米或英里表示的位置。显然，我们可以对这些维度进行降维操作，因为不同的速度变量或位置变量给出的信息相同（相关变量），因此相应的子空间可以由两个不相关的维度组成（速度变量和位置变量）。PCA 不仅寻找不相关变量，还寻找方差最大的维度。也就是说，对于用 km/h 和 m/s 表示的速度，该算法选择方差较大的变量，这两种速度表示之间的关系为 velocity[km/h]=3.6*velocity[m/s]，函数图像见图 2.6（表示这种函数关系的直线更靠近 km/h 轴，因为 1km/h=3.6m/s）。速度在 km/h 轴上的投影（projection）比起在 m/s 轴上的更加分散）。

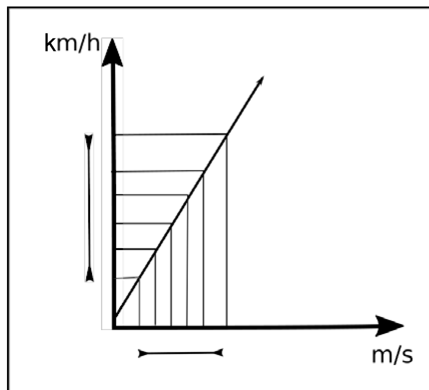


图 2.6 用 m/s 和 km/h 表示的速度之间的线性函数关系

图 2.6 为用 m/s 和 km/h 表示的速度两者之间为线性函数关系。数据点在 km/h 轴上的投影方差较大，而 m/s 轴上的投影方差较小。沿直线 velocity [km/h]= 3.6*velocity[m/s]分布的数据点的方差要大于两条轴上投影的方差。

有了上述例子作铺垫，现在我们可以详细讨论这种方法及其特点。不难看出，寻找方差最大化的不相关维度等价于按以下步骤进行计算。同样，假定我们有特征向量 $x^{(i)}$，$l \in 1,\cdots,N$：

- 数据集的均值：$\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$
- 均值发生漂移后得到的数据集：$u^{(i)} = x^{(i)} - \mu, i \in 1,\cdots,N$
- 调整后的数据集，其中每个特征向量的分量 $u_j^{(i)}$ 除以标准差：$u_j^{(i)} / \sigma_j \to u_j^{(i)}$，

$$\sigma_j = \left(\frac{1}{N} \sum_{i=1}^{N} \left(x_j^{(i)} \right)^2 \right)^{1/2}$$ ①

① 公式大括号里面的系数为 1/N。——译者注

- 样本点的协方差矩阵：$\Sigma = \frac{1}{N-1}\sum_{i=1}^{N}\left(u^{(i)}\right)^{T} u^{(i)}$
- k 个最大的特征值（eigenvalue）λ_i，$i \in 1,\cdots,k$，以及相应的特征向量（eigenvector）$w^{(i)}$，$i \in 1,\cdots,k$。
- 映射到由 k 个特征向量组成的子空间的特征向量 $v^{(i)} = W^{T} u^{(i)} \in R^{k}$，其中，$W$=[$w^{1}\cdots w^{k}$] $\in R^{Nxk}$ 是 N 行 k 列的特征向量矩阵。

最终，特征的向量（主成分）$v^{(i)}$ 位于子空间 R^{k}，它仍然保留着原始向量的最大方差（和信息）。

处理高维数据集，比如脸部识别任务，PCA 降维技术非常有用。对于脸部识别任务，需要比对输入的图像和数据库中的其他图像，以找到正确的人。用 PCA 实现的 **Eigenfaces**（特征脸）应用，利用每张图像中大量像素是相关的这一特点实现人脸识别。例如，图像背景处的像素都是相关（相同）的，因此可以用降维方法，在维度更少的子空间比较图像，可以更快地给出准确结果。作者的 GitHub 主页放了 Eigenfaces 的一种实现方式，详见 https://github.com/ai2010/eigenfaces。

PCA 示例

下面这个例子讲解 PCA 和 NumPy 库的用法，后者我们在第 1 章介绍过。我们的任务是识别二维数据集的主成分。该数据集中的数据点沿直线 y=2x 分布，其中包含一些随机（正态分布）添加的噪声。数据集及其图像（见图 2.7）用以下代码生成：

```
import numpy as np
from matplotlib import pyplot as plt

#line y = 2*x
x = np.arange(1,101,1).astype(float)
y = 2*np.arange(1,101,1).astype(float)
#add noise
noise = np.random.normal(0, 10, 100)
y += noise

fig = plt.figure(figsize=(10,10))
#plot
plt.plot(x,y,'ro')
plt.axis([0,102, -20,220])
plt.quiver(60, 100,10-0, 20-0, scale_units='xy', scale=1)
plt.arrow(60, 100,10-0, 20-0,head_width=2.5, head_length=2.5, fc='k',
```

```
ec='k')
plt.text(70, 110, r'$v^1$', fontsize=20)

#save
ax = fig.add_subplot(111)
ax.axis([0,102, -20,220])
ax.set_xlabel('x',fontsize=40)
ax.set_ylabel('y',fontsize=40)
fig.suptitle('2 dimensional dataset',fontsize=40)
fig.savefig('pca_data.png')
```

图 2.7 就是我们生成的数据集。显然，数据点沿一定方向分布，它对应的是我们要从数据中抽取的主成分 V^1。

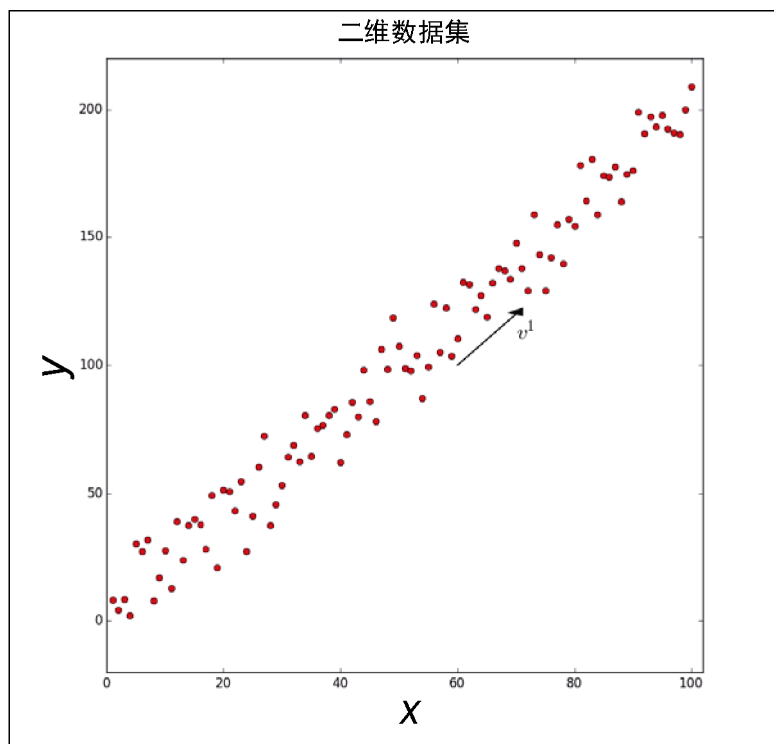


图 2.7 二维数据集。主成分的 v^1 方向用箭头表示

该算法计算二维数据集的均值和均值漂移后得到数据集的均值，然后用相应的标准差调整数据的取值范围：

```
mean_x = np.mean(x)
```

```
mean_y = np.mean(y)
u_x = (x- mean_x)/np.std(x)
u_y = (y-mean_y)/np.std(y)
sigma = np.cov([u_x,u_y])
```

为了抽取主成分，我们需要计算特征值和特征向量，选择特征值最大的特征向量：

```
eig_vals, eig_vecs = np.linalg.eig(sigma)
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i])
              for i in range(len(eig_vals))]

eig_pairs.sort()
eig_pairs.reverse()
v1 = eig_pairs[0][1]
print v1
array([ 0.70710678, 0.70710678]
```

为了确认主成分是否按照预期沿直线分布，我们需要将其坐标调整回去：

```
x_v1 = v1[0]*np.std(x)+mean_x
y_v1 = v1[1]*np.std(y)+mean_y
print 'slope:',(y_v1-1)/(x_v1-1)
slope: 2.03082418796
```

计算得到斜率约为 2，这跟我们一开始选取的数值一致。scikit-learn 库提供了一种快捷的 PCA 算法实现，无须调整数据取值范围或移动均值。使用 sklearn 模块之前，我们需要将调整后的数据转换为矩阵格式，矩阵的每一行为用 x、y 坐标表示的数据点：

```
X = np.array([u_x,u_y])
X = X.T
print X.shape
(100,2)
```

现在可以运行 PCA 模块，指定目标主成分数量（该例中我们将其设置为 1）：

```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
pca.fit(X)
v1_sklearn = pca.components_[0]
print v1_sklearn
[ 0.70710678 0.70710678]
```

用这种方法得到的主成分[0.70710678 0.70710678]与我们前面一步步计算得到的完

全相同，因此直线的斜率也相同。现在，我们可以用这两种方法将数据集转换到新的一维空间：

```
#transform in reduced space
X_red_sklearn = pca.fit_transform(X)
W = np.array(v1.reshape(2,1))
X_red = W.T.dot(X.T)
#check the reduced matrices are equal
assert X_red.T.all() == X_red_sklearn.all(), 'problem with the pca
algorithm'
```

assert 断言语句没有抛出异常，因此两种方法的结果完全一致。

2.3 奇异值分解（SVD）

该方法是基于定理：矩阵 $X\ dxN$ 可以分解为

$$X = U \Sigma V^T$$

其中：

- U 是一个 dxd 的酉矩阵。
- Σ 是一个 dxN 的对角矩阵，对角线元素 σ_i 叫作奇异值。
- V 是一个 NxN 酉矩阵。

该例，X 由特征向量 $x^{(i)}\ i \in 1, \cdots, N$ 构成，其中 $x^{(i)} \in R^d$ 表示一列。我们可以减少每个特征向量的维数 d，去近似奇异值分解。在实际操作中，我们只考虑最大的奇异值 σ_1， σ_t，$t<d$，使得

$$X \simeq U_t \Sigma_t V_t^T, U_t\left(dxt\right), \Sigma_t\left(txt\right), V^T\left(txN\right)$$

t 表示降维后得到的空间维度，该空间也就是特征向量映射到的空间。向量 $x^{(i)}$ 按如下的公式转换到新空间：

$$x_t^{(i)} = \left(x^{(i)}\right)^T U_t \Sigma_t^{-1} \in R^t$$

这表明，矩阵 $\Sigma_t V_t^T$ （而不是 V_t^T ）表示在 t 维空间的特征向量。

该方法与 PCA 非常相似；实际上，scikit-learn 库正是用 SVD 算法来实现 PCA。

2.4　小结

本章详细讨论了主要的聚类算法。我们实现了这些算法（用 scikit-learn）并比较了它们结果的好坏。我们还讲解和实现了最主要的降维技术——主成分分析法（PCA）。学完本章，你应该已具备用 Python 及其库实现无监督学习算法，解决实际问题的能力。

在下一章中，我们将讨论分类和回归这两类有监督学习算法。

第 3 章
有监督机器学习

本章讨论最常用的回归和分类技术。这类算法背后的机制是相同的。通常而言，有监督学习算法指的是分类和回归。本章，我们会依次讨论线性回归、朴素贝叶斯、决策树和支持向量机算法。我们将用这些算法解决一个分类问题和一个回归问题，以帮助你理解它们的使用方法。前言部分也曾讲过，有监督学习要用标注好的训练集训练模型，找到合适的参数值。跟之前一样，本章代码也已放到我的 GitHub 主页本章文件夹中，地址是 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_3/。

本章最后，我们将介绍另一种也可以实现分类的算法（隐马尔可夫模型），虽然它不是专门用来处理分类问题的。我们现在先来解释这些方法在预测数据集标签这类问题上常见的出错原因。

3.1　模型错误评估

我们前面讲过，用训练好的模型去预测新数据的标签，预测结果的质量取决于模型的泛化能力，即正确预测在训练数据中未出现的数据的能力。该问题的研究文献很多，一般涉及两个概念：输出的偏差（bias）和方差（variance）。偏差是指由算法的错误假设导致的错误。给定标签为 y_t 的数据点 $x^{(t)}$，如果用不同的训练集训练，模型就会有偏差，预测结果 y_t^{pred} 将总是不同于 y_t。而方差误差（variance error）是指对于给定数据点 $x^{(t)}$，给出了不同的、错误的预测标签。举个典型的例子，假定有一组同心圆，正确的标签值（实际标签）位于中心区域，如图 3.1 所示。预测结果越接近中心，模型偏差和方差越小（图 3.1 左上方位置）。其他 3 种情况一并显示如下。

方差和偏差较低的模型，用小点（见图 3.1）表示的预测结果集中分布在红心[①]位置（实际标签）。偏差较大的模型，预测结果远离实际标签所在位置，而方差较高的模型，预测结

① 同心圆中最小的那一个。——译者注

果分布范围较广。

标签可以是连续型或离散型，分别对应回归和分类问题。大多数模型适用于这两类问题。下文中的回归和分类这两个词指同一个模型。更加正式地讲，给定一个由 N 个数据点组成的数据集和这些数据点对应的标签 y_t，$t\in 1,\cdots,N$，模型的参数为 $\theta=\theta_0;$，θ_{M-1}，或表示为 θ_j，$j\in 0,\cdots, M-1$，并且，参数的实际估计值（true parameter）为 $\hat{\theta}=\hat{\theta}_0$，$\cdots$，$\hat{\theta}_{M-1}, \hat{\theta}_j$，$j\in 0,\cdots, M-1$，那么模型的**均方误差**（Mean Square Error，MSE）等于：

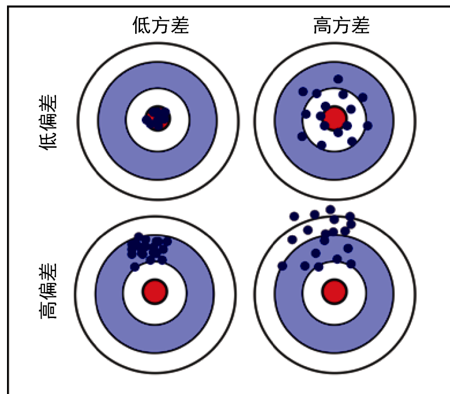


图 3.1 方差和偏差示例图

$$MSE(\hat{\theta})=\frac{1}{N}\sum_{t=1}^{N}\left(y_t-y_t^{pred}\right)^2=E\left[(\hat{\theta}-\theta)\right]=E\left[(\theta-E(\theta))^2\right]+(E(\theta)-\hat{\theta})^2=Var(\theta)+Bias(\theta,\hat{\theta})^2$$

我们将用均方误差这一评价指标来评估本章所讲算法的好坏。我们现在开始讲解广义线性方法。

3.2 广义线性模型

广义线性模型是一类模型的统称，这类模型尝试寻找 M 个使标签 y_i 和特征向量 $x^{(i)}$ 形成线性关系的参数 θ_i，$j\in 0,\cdots, M-1$，如下所示：

$$y_i=\sum_{j=0}^{M-1}\theta_j x_j^{(i)}+\in_i=h_\theta(x^{(i)})+\in_i \quad \forall i\in 0,\cdots,N-1$$

其中，$\in_i$ 指模型的错误。算法在寻找参数值时，尝试最小化由代价函数（cost function[①]）所定义的模型的错误 J：

$$J=\frac{1}{2}\sum_{i=0}^{N-1}\left(y_i-h_\theta(x^{(i)})\right)^2$$

J 的最小化用迭代算法**批量梯度下降**（batch gradient descent）实现：

$$\theta_j\leftarrow\theta_j+\alpha\sum_{i=0}^{N-1}\frac{\partial J}{\partial\theta_j},\forall j\in 0,\cdots,M-1$$

① 又称为 distortion function、loss function。——译者注

上式中的 α 叫作学习速率（learning rate），它是收敛速度和收敛精度之间的折衷。另一种算法叫作**随机梯度下降**（stochastic gradient descent），对 $i \in 0, \cdots, N\text{-}1$ 进行迭代：

$$\theta_j \leftarrow \theta_j + \alpha \frac{\partial J}{\partial \theta_j}, \forall j \in 0, \cdots, M-1$$

对于每个训练样例 i，更新 θ_j，而不是等到最后对训练集全部样例的梯度加总后再更新 θ[①]。随机梯度下降算法收敛至 J 的最小值附近，它通常比批量梯度下降算法收敛速度更快，但是随机梯度下降最后得到的结果在实际参数值附近波动。下一节介绍最常用的模型 $h_\theta(x^{(i)})$ 和相应的代价函数 J。

1．线性回归

线性回归是最简单的回归算法，它基于以下模型：

$$h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots = \sum_{j=0}^{M-1} \theta_j x_j^{(i)}, \forall j \in 0, \cdots, N-1$$

它的代价函数和更新规则如下：

$$J = \frac{1}{2}\sum_{i=0}^{N-1}\left(y_i - h_\theta\left(x^{(i)}\right)\right)^2 \rightarrow \frac{\partial J}{\partial \theta_j} = \left(y_i - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)} \forall j \in 0, \cdots, M-1$$

2．岭回归

岭回归（ridge regression）也叫作**吉洪诺夫正则化**（Tikhonov regularization），它在代价函数 J 中增加了正则项：

$$J = \frac{1}{2}\sum_{i=0}^{N-1}\left(y_i - h_\theta\left(x^{(i)}\right)\right)^2 + \frac{\lambda}{2}\sum_{j=0}^{M-1}\theta_j^2 \rightarrow \frac{\partial J}{\partial \theta_j} = \left(y_i - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)} + \lambda\theta_j$$

$\forall j \in 0, \cdots, M\text{-}1$，$\lambda$ 为正则化参数。新增的正则项，其功能是在所有可能的参数中，增加一组特定参数的权重，惩罚所有值不等于 0 的参数 θ_j，使得最终得到的参数 θ_j 收缩（shrink）于 0，这样做可以降低参数的方差，但是引入了偏差。线性回归的参数用带上标的 l 表示，岭回归的参数与线性回归的参数，两者之间的关系表示如下：

$$\theta_j = \frac{\theta_j^l}{1+\lambda}$$

显而易见，λ 的值越大，岭回归参数越向 0 收缩。

① 原文为“instead of waiting to sum over the entire training set”。——译者注

3．套索回归

套索回归（lasso regression）算法与岭回归类似，唯一的区别是，套索回归的正则项为所有参数绝对值之和：

$$J=\frac{1}{2}\sum_{i=0}^{N-1}\left(y_i-h_\theta\left(x^{(i)}\right)\right)^2+\lambda\sum_{j=0}^{M-1}\left|\theta_j\right|\rightarrow\frac{\partial J}{\partial\theta_j}=\left(y_i-h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}+\lambda sign(\theta_j)$$
$$\forall j\in 0,\cdots,M-1$$

4．对数几率回归

对数几率回归（logistic regression）算法适用于（二值）分类问题。因此，我们把标签定义为 $y_i\in 0,1$。模型用以下对数几率函数来表示：

$$h_\theta\left(x^{(i)}\right)=\frac{1}{1+e^{-\sum_{j=0}^{M-1}\theta_j x_j^{(i)}}}$$

它的代价函数定义如下：

$$J=\frac{1}{2}\sum_{i=0}^{N-1}y_i\log\left(h_\theta\left(x^{(i)}\right)\right)+(1-y_i)\log\left(1-h_\theta\left(x^{(i)}\right)\right)$$

更新权重的规则，其形式与线性回归的相同（但模型的定义 h_θ 不同）：

$$\frac{\partial J}{\partial\theta_j}=\left(y_i-h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}\quad\forall j\in 0,\cdots,M-1$$

注意，数据点 p 的预测结果 $h_\theta(x^{(p)})$ 是一个介于0到1之间的连续型值。因此，为了预测类别标签，我们通常将 $h_\theta(x^{(p)})$=0.5 设置为阈值，阈值跟标签对应关系如下：

$$h_\theta(x^{(p)})=\begin{cases}\geqslant 0.5 & 1\\ <0.5 & 0\end{cases}$$

对数几率回归算法用一对多（one versus all）或一对一技术（one versus one），可实现多个类别的分类问题。一对多方法，包含 K 个类别的分类问题，可通过训练 K 个对数几率回归模型来解决。处理类别 j 的模型，将类别 j 看作为+1，剩余类别看作0[①]；一对一方法，需要为任意两个类别训练一个模型（共是 $\frac{K(K-1)}{2}$ 个训练模型）。

---

① 一对多方法，用 K 个模型处理被预测数据，得到 K 个结果之后，从中选取预测值最高的作为被预测数据的类别。一对一方法，根据少数服从多数的原则，从 $\frac{K(K-1)}{2}$ 个模型的预测结果中选择出现次数最多的类别，作为最终的类别。——译者注

3.2.1 广义线性模型的概率解释

前面介绍了广义线性模型，现在我们来寻找满足如下关系的参数θ_j：

$$y_i = \sum_{j=0}^{M-1} \theta_j x_j^{(i)} + \epsilon_i = h_\theta\left(x^{(i)}\right) + \epsilon_i \ \forall i \in 0, \cdots, N-1$$

对于线性回归，我们可以假定ϵ_i服从均值为 0、方差为 σ^2 的正态分布，其概率 $p(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\epsilon_i^2}{2\sigma^2}}$ 等价于：

$$p\left(y_i x^{(i)}; \theta\right) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\left(y_i - h_\theta\left(x^{(i)}\right)\right)^2}{2\sigma^{\wedge 2}}}$$

因而，系统的总似然度可以表示为：

$$L(\theta) = \prod_{i=0}^{N-1} p\left(y_i \middle| x^{(i)}; \theta\right) = \prod_{i=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\left(y_i - h_\theta\left(x^{(i)}\right)\right)}{2\sigma^2}}$$

对数几率回归算法，我们假定对数几率函数自身就是概率：

$$P\left(y_i = 1 \middle| x^{(i)}; \theta\right) = h_\theta(x^{(i)})$$

$$P\left(y_i = 0 \middle| x^{(i)}; \theta\right) = 1 - h_\theta(x^{(i)})$$

那么，似然度可以表示为：

$$L(\theta) = \prod_{i=0}^{N-1} p\left(y_i \middle| x^{(i)}; \theta\right) = \prod_{i=0}^{N-1} \left(h_\theta\left(x^{(i)}\right)\right)^{y_i} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{(1-y_i)}$$

上述两种情况，最大化似然度等价于最小化代价函数，因此用相同的梯度下降方法求解。

3.2.2 k 近邻

k 近邻（k-nearest neighbours，KNN）是一种非常简单的分类（或回归）方法。给定一组特征向量$x^{(i)}$ $i \in 1, \cdots, N$ 及相应的标签y_i，待分类的数据点$x^{(t)}$的 K 个近邻为$x^{(k)}$ $k \in 1$, …, K。k 近邻算法将近邻的多个标签中出现次数最多的分配给$x^{(t)}$。寻找近邻时，常用的距离度量方法有：

- 欧氏距离（Euclidean）：$\sqrt{\sum_{j=0}^{M-1} \left(x_j^{(k)} - x_j^{(t)}\right)^2}$

- **曼哈顿距离**（Manhattan）：$\left|\sum_{j=0}^{M-1} x_j^{(k)} - x_j^{(t)}\right|$

- **闵氏距离**（Minkowski）：$\left(\sum_{j=0}^{M-1}\left|x_j^{(k)} - x_j^{(t)}\right|^q\right)^{1/q}$　（如果 q=2，即为欧氏距离）

若是回归问题，计算 y_t，需要将近邻出现次数最多的标签 y_k，$k \in 1, \cdots, K$ 替换为近邻标签的均值。最简单的均值（或大多数情况对应哪个标签）形式是各近邻的权重相同，因此每个数据点，不管它与 $x^{(t)}$ 实际距离有多近，重要程度相同。然而，也可以使用近邻与 $x^{(t)}$ 之间的距离倒的加权平均值。

3.3　朴素贝叶斯

朴素贝叶斯（Naive Bayes）分类算法是基于贝叶斯概率定理和各特征之间相互独立的假设。给定 m 个特征 x_i，$i \in 0, \cdots, M$-1 和一组标签（类别）$y \in 0, \cdots, K$-1，根据贝叶斯定理，标签 c（给定特征集 x_i）的概率为：

$$P\left(y=c \middle| x_{0.}.,x_{M-1}\right) = \frac{P\left(x_{0.}.,x_{M-1} \middle| y=c\right)P(y=c)}{P(x_{0.}.,x_{M-1})}$$

其中：

- $P(x_0,.,x_{M-1}|y=c)$叫作似然分布；
- $P(c|x_0,.,x_{M-1})$为后验分布；
- $P(y=c)$为前验分布；
- $P(x_0,.,x_{M-1})$叫作证据（evidence）。

预测特征 x_i，$i \in 0, \cdots, M-1$ 所属的类别，即是求取使得概率 p 取得最大值的标签：

$$P = arg \max_{y \in 0, \cdots, K-1} P\left(y \mid x_{0,.}.,x_{M-1}\right)$$

然而，上面这个式子无法直接计算。因此，需要做出假设。

用条件概率公式 $P(A|B)=\frac{P(A,B)}{P(B)}$，我们可以将前面那个公式的分子写成：

$$P(x_{0,}.,x_{M-1}|y=c)P(y=c) = P(y=c)P(x_0|y=c)p(x_{1,}.,x_{M-1}|y=c,x_0) =$$
$$P(y=c)P(x_0|y=c)P(x_1|y=c,x_0)\,p(x_{2,}.,x_{M-1}|y=c,x_0,x_1) =$$

$$P(y=c)P(x_0|y=c)P(x_1|y=c,x_0)\,p(x_{M-1}|y=c,x_0,x_{1,}..,x_{M-2})$$

我们的假设是，给定标签 c，各个特征 x_i 是之间相互独立的（例如，要计算 x_1 标签为 c 的概率，只要有标签 c 的知识就足够了，特征 x_0 的相关知识是多余的，$P(x_1|y=c)=P(x_1|y=c,\ x_0)$）：

$$P(y=c)=\prod_{j=0}^{M-1}P(x_j|y=c)$$

在该假设下，y 为标签 c 的概率等于：

$$P(y=c|x_{0,..},x_{M-1})=\frac{\prod_{j=0}^{M-1}P(x_j|y=c)P(y=c)+1}{\sum_{i=0}^{K-1}\prod_{j=0}^{M-1}P(x_j|y=i)P(y=i)+M} \quad (1)$$

其中，分子中+1 和分母中的 M 为常数项，使用它们是为了避免出现 0/0 情况（**拉普拉斯平滑**）。

由于（**1**）式中的分母对于所有的标签都是相同的（将标签所有可能情况加起来），因此最终类别的概率，是通过最大化中（**1**）式的分子得到的：

$$p=arg\max_{c\in 0,\cdots,K-1}\prod_{j=0}^{M-1}P(x_j|y=c)P(y=c) \quad (2)$$

对于我们一直使用的训练集 $x^{(i)}$，$i\in 0,\cdots,\ N-1$，其中 $x^{(i)}\in R^M$（M 个特征）和相应的标签集 y_i，$i\in 0,\ \cdots,N-1$，$P(y=c)$的概率是通过计算训练样例中，标签为 c 的样例的频数，即 $P(y=c)=\frac{N_{y=c}}{N}$。条件概率 $P(x_j|y=i)$则需要根据某种分布进行计算。我们接下来讨论两种模型：**多项式朴素贝叶斯**（Multinomial Naive Bayes）和**高斯朴素贝叶斯**（Gaussian Naive Bayes）。

3.3.1 多项式朴素贝叶斯

假定我们想判断一封由一组词 $x^{(s)}=(x_0^{(s)},\cdot\cdot,x_{M-1}^{(s)})$ 组成的邮件 s 是否是垃圾邮件，如果是则为 1，不是为 0，因此 $y\in 0,1$。M 为词汇量（特征数）多少。一共有 w_t 个词，$t\in 0,\ ..,\ M-1$ 个词，N 个训练样例（邮件）。

对于每一封标签为 y_i 的邮件 $x^{(i)}$。$x_j^{(i)}, j\in 0,..,M-1$ 表示单词 j 出现在训练样例 i 中的次数。例如，$x_1^{(3)}$ 表示单词 1 或 w_1 出现在第 3 封邮件中的次数。我们采用多项式分布的似然度：

$$p\left(x^{(s)}\middle|y\right)=\frac{\left(\sum_{j=0}^{M-1}x_j^{(s)}\right)!}{\prod_{j=0}^{M-1}x_j^{(s)}!}\prod_{j=0}^{M-1}P\left(w_j\middle|y\right)^{x_j^{(s)}}\alpha\prod_{j=0}^{M-1}P\left(w_j\middle|y\right)^{x_j^{(s)}}$$

上式，前面的归一化常数（normalization constant）可以省略，因为它们不依赖于标签 y，所以不会影响 *arg max* 运算符的运算结果。重要的是计算单词 w_j 在整个训练集上的概率：

$$p\left(w_j\middle|y\right)=\frac{\sum_{i=0}^{N-1}x_j^{(i)}\delta_{y_i,y}}{\sum_{t=0}^{M-1}\sum_{i=0}^{N-1}x_t^{(i)}\delta_{y_i,y}}=\frac{N_{jy}}{N_y}$$

上式，N_{jy} 表示标签为 y 的单词 j 的出现次数。N_y 表示训练集中标签为 y 的邮件数。

上式相当于计算（**1**）式中的 $P(x_j|y=i)$和 $P(x_j|y=c)$，也就是计算多项式分布的似然度。由于概率的指数运算结果很小，可能会导致数值下溢，因此算法最后一步（**2**）常采用对数概率形式：

$$p=arg\max_{y}logP(y)-\sum_{j=0}^{M-1}x_j^{(s)}logP(w_j|y)$$

3.3.2　高斯朴素贝叶斯

特征向量 $x^{(i)}$ 若取连续值，则可以使用高斯朴素贝叶斯。例如，我们想将图像分为 K 个类别，每个特征 j 为一个像素。$x_j^{(i)}$ 表示训练集第 i 张图像的第 j 个像素。训练集共有 N 张图像，其类别为 y_i, $i\in 0,\cdots, N-1$。给定一张用像素 $x_0,\cdots,x_{M-1}$ 表示的、未标记的图像，公式（**1**）中的 $P(x_j|y=i)$变为：

$$p\left(x_j\middle|y=i\right)=\frac{1}{\sigma_{ji}\sqrt{2\pi}}e^{-\frac{\left(x_j-\mu_{ij}\right)^2}{2\sigma_{ji}^2}}$$

其中，均值为

$$\mu_{ij}=\frac{\sum_{t=0}^{N-1}x_j^{(t)}\delta_{y_t,i}}{\sum_{t=0}^{N-1}\delta_{y_t,i}}$$

方差为

$$\sigma_{ij}^2 = \frac{\sum_{t=0}^{N-1}\left(x_j^{(t)} - \mu_{ij}\right)^2 \delta_{y_t,i}}{\sum_{t=0}^{N-1}\delta_{y_t,i} - 1}$$

3.4 决策树

该类算法从特征值中学习，生成一套简单的规则来分割训练集，预测未知的标签。例如，根据湿度、风力、气温和气压的值，判断今天是否需要带伞，这是一个分类问题。根据过去 100 天的数据，我们可以给出下面这样一个决策树示例图。样表见表 3.1（1mbar=100Pa）：

表 3.1

湿度/%	气压/mbar	风力/(km/h)	气温/℃	是否带伞
56	1021	5	21	是
65	1018	3	18	否
80	1020	10	17	否
81	1015	11	20	是

在图 3.2 中，小矩形中的数字表示有多少天带伞，椭圆中的数字表示有多少天没带伞。

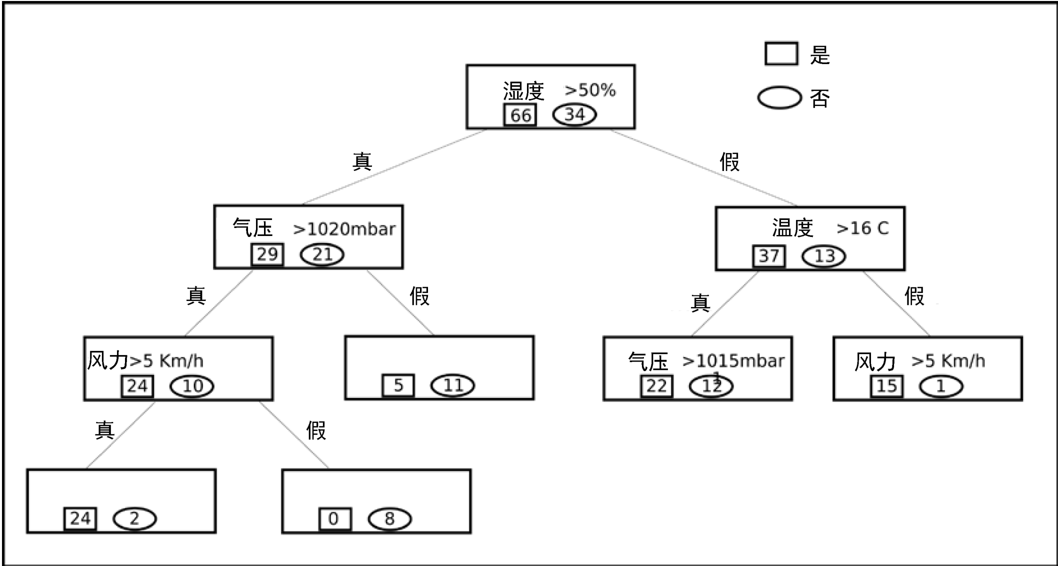


图 3.2 根据过去 100 天的数据记录，预测是否需要带伞的决策树

决策树有两种类型的节点：

（1）决策节点，根据决策切分数据集，得到两个（或更多）分枝；

（2）叶子节点，数据完成分类后得到这类节点。

通常用最大决策节点数量（树的深度）或继续切分所需最少数据点数量（常用 2 到 5）作为退出机制。决策树学习的问题是，如何在所有可能的节点组合中，构造一棵最佳的树，也就是估计所要采用的规则的层次(换句话说,第 1 个节点用湿度还是气温,以及后续节点选用什么规则)。更正式地讲，给定训练集 $x^{(i)}, i \in 1,\cdots,N$，其中 $x^{(i)} \in R^m$，标签为 y_i。我们需要找出在节点 k 能够为数据 S 做出最佳切分的规则。如果选定的特征 j 为连续值，每条切分规则用特征 j 和阈值 t_k^j 表示，若 $x_j^{(i)} < t_k^j$，将 S 分到 $S_{left}(t_k^j, j)$；若 $x_j^{(i)} \geqslant t_k^j$ 则分到 $S_{right}(t_k^j, j)$，其中，$i \in 1, \cdots, N$。节点 k 的最佳切分规则 (t_k^q, q) 对应的是 I 函数的最小值，也就是最小不纯度。I 函数用来度量切分规则在多大程度上能够将数据分到标签不同的分枝中去（也就是说，每枝数据的标签混合程度最小）：

$$I(t_k^j, j) = \frac{n_{left}}{N_k} H(S_{left}) + \frac{n_{right}}{N_k} H(S_{right})$$

$$\left(t_k^q, q\right) = arg \min_{\left(t_k^j, j\right)} I\left(t_k^j, j\right)$$

其中，n_{left} 和 n_{right} 分别表示左枝和右枝数据点数量。N_k 表示在节点 k 处，数据点的数量。假定用 b（b 可以是左或右分枝）分枝每个目标值 l 的概率 $p_{bl} = \frac{n_l}{n_b}$ 来表示，度量方法 H 可以有不同的形式：

- 分枝的熵：$H(S_b) = \sum_l p_{bl} \log_2 p_{bl}$
- 分枝的基尼不纯度（Gini impurity）：$H(S_b) = \sum_l p_{bl}(1 - p_{bl})$
- 分类错误：$H(S_b) = 1 - \max_l \left(p_{bl}\right)$
- 均方误差（方差）：$H(S_b) = \frac{1}{N_b} \sum_{i \in N_b} (y_i - \mu_b)^2$, 其中 $\mu_b = \frac{\sum_{i \in N_b} y_i}{N_b}$

请注意，后者常用于回归问题，而前两个多用于分类问题。还要注意，研究文献中常引入信息增益（information gain）这个定义，来表示节点 k 的 H 值和 $I(t_k^j ,j)$之间的差距：

$$IG = H(S_k) - I(t_k^j, j)，\text{其中 } H(S_k) = \sum_l p_{kl} \log_2 p_{kl}, p_{kl} = \frac{n_l}{N_k}$$

如果特征 j，有 d 个可能的离散值，也就不存在将其分到两个类别的阈值 t_k^j。我们要将

数据分到 d 个类别中的一个，需要计算 d 个类别每个类别的 H 值。

例如，我们可以用熵作为不纯度的度量标准 H，来确定第 1 个节点（k=0）的切分规则。

如果所有特征的取值为连续型值，那么就需要 t_0^j 。假定 j=0 为湿度，将数据集中可能的湿度值按升序排列，如表 3.2 所示。

表 3.2

h	0		1		…		98		99	
带伞	yes		no		…		no		no	
湿度	58		62		…		88		89	
	<	>=	<	>=	<	>=	<	>=	<	>=
是	0	11	14	32	7	20	29	12	78	0
否	0	89	21	33	13	60	10	49	22	0
$I(x_0^{(h)}, 0)$	0.5		0.99		0.85		0.76		0.76	

因此，湿度这一特征的阈值为 $t_0^0 = arg \min\limits_{h\in 1,\cdots,N}\left(I\left(x_j^{(h)}, j\right)\right) = 58$。我们可以按照相同的方式计算气温 t_0^1 、风力 t_0^2 和气压 t_0^3 的阈值。计算完成后，我们就可以计算 4 个特征每个特征的不纯度，以决定第 1 个节点的最佳切分规则，见表 3.3。

表 3.3

是		带伞		是 否		带伞	
		否					
湿度 j=0	$x_0^{(t)} < t_0^0$	0	0	气温 j=1	$x_0^{(t)} < t_0^1$	21	32
	$x_0^{(t)} \geqslant t_0^1$	11	89		$x_0^{(t)} \geqslant t_0^1$	11	36
不纯度：$I(t_0^0,0)$=0.5				不纯度：$I(t_0^1,1)$=0.88			
是		带伞		是 否		带伞	
		否					
风力 j=2	$x_0^{(t)} < t_0^2$	48	5	气压 j=3	$x_0^{(t)} < t_0^3$	39	3
	$x_0^{(t)} \geqslant t_0^2$	1	46		$x_0^{(t)} \geqslant t_0^3$	45	13
不纯度：$I(t_0^2,2)$=0.31				不纯度：$I(t_0^3,3)$=0.60			

因而，节点 0 处，最佳切分规则为：

$$\left(t_0^q, q\right) = arg \min_{j=0,1,2,3}\left(I\left(t_0^0, 0\right), I\left(t_0^1, 1\right), I\left(t_0^2, 2\right), I\left(t_0^3, 3\right)\right) = \left(t_0^2, 2\right)$$

也就是阈值为 t_0^2 的风力。我们可以沿着树往下走，重复使用相同的方法，找出其余决策节点的最佳切分规则，直接到达叶子节点。

决策树学习，能够处理大型数据集，尽管它的泛化能力不是很好，尤其是特征数量很大时（$N \approx M$）。遇到这种情况，建议使用深度较小的树，或使用降维技术。设定进一步切分所需最少数据点数量，或设定叶子节点的最小数量，有助于防止出现过拟合问题（overfitting）。决策树算法可能会生成过于复杂的树；这时可采用剪枝（pruning）技术，去除不影响预测质量的分枝。剪枝技术有多种，但是它们超出了本书的讲解范围。请注意，我们还可以同时训练多棵决策树，组成所谓的**随机森林**（random forest）。随机森林算法，从原始数据集随机选取一部分数据点训练一棵树，该树每个决策节点的学习，也使用随机选取的特征子集。回归问题，取多棵决策树预测结果的均值作为最终预测结果；而分类问题，则按照少数服从多数原则选取最终预测结果。

3.5　支持向量机

支持向量机（Support Vector Machine，SVM），算法尝试从几何视角将数据集 $x^{(i)}$ $i \in 1,\cdots,N$ 分成标签为 y_i=+1 和 y_i=-1 的两个子集。在图 3.3 中，数据被完美地分成两个类别（空心圆和实心圆），也就是说由实线表示的决策边界（或超平面）完全将两个类别分开（换句话说，没有分错类的数据点）：

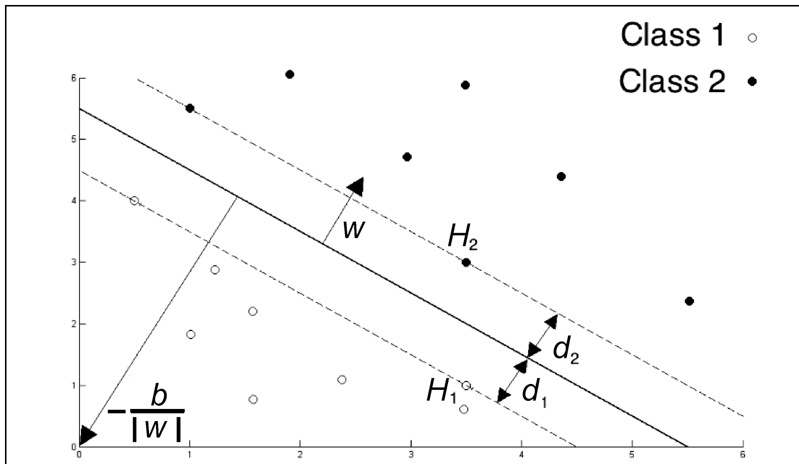


图 3.3　实线（决策边界）将数据集分为两类（空心和实心圆）

超平面用数学语言可表示为等式 $w \cdot x + b = 0$，$|x| = \dfrac{-b}{|w|}$ 为超平面与原点之间的距离，w 为超平面的法线。支持向量机算法的目标是，最大化各数据点到决策边界之间的距离。实际操作中，我们仅考虑距离决策边界最近、叫作支持向量的数据点 i。它们位于平面 H_1 和 H_2 上，与决策边界之间的距离分别为 d_1 和 d_2：

H_1超平面：$w\cdot x^{(i)}+b=+1$，y_i=+1 --------（1）

H_2超平面：$w\cdot x^{(i)}+b=-1$，y_i=−1 --------（2）

超平面与决策边界之间的距离叫作间隔（margin），假定$d_1=d_2$，那么，支持向量机方法要找的就是使得间隔最大的 w 和 b 的值。

因为H_1和H_2之间的距离为$\frac{2}{|w|}$，所以间隔为$\frac{1}{|w|}$，那么支持向量机算法等价于求解如下问题：

$$min\frac{1}{2}|w|^2 \text{ such that } y_i\left(w,x^{(i)}+b\right)-1\geqslant 0\ \forall i\in 1,..,N$$

为了使用二次规划方法求解这个数学问题，我们在上式中添加了平方运算和因子 1/2。然后，我们使用拉格朗日乘子法将这个问题改写为如下拉格朗日函数，其中α_i>0 为拉格朗日乘子：

$$L=\frac{1}{2}|w|^2-\sum_{i=0}^{N-1}\alpha_i\left[y_i\left(x^{(i)}\bullet w+b\right)-1\right]=\frac{1}{2}|w|^2-\sum_{i=0}^{N-1}\alpha_i y_i\left(x^{(i)}\bullet w+b\right)+\sum_{i=0}^{N-1}\alpha_i$$

分别对|w|和|b|求偏导数，并令导数为 0，我们得到：

$$w=\sum_{i=0}^{N\text{-}1}\alpha_i y_i x^{(i)} \qquad (3)$$

$$\sum_{i=0}^{N\text{-}1}\alpha_i y_i=0 \qquad (4)$$

优化过后的拉格朗日函数为：

$$L_d=\sum_{i=0}^{N-1}\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i H_{ij}\alpha_j\alpha_i\geqslant 0\forall i\in 0,..,N-1,\sum_{i=0}^{N-1}y_i\alpha_i=0$$

其中，$H_{ij}=y_i y_j x^{(i)}\cdot x^{(j)}$。

上式称为原问题的对偶问题，这时我们只要找到α_i的最大值即可：

$$\max_{\alpha_i}\left[\sum_{i=0}^{N-1}\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i H_{ij}\alpha_j\right]a_i\geqslant 0\forall i\in 0,..,N-1,\sum_{i=0}^{N-1}y_i\alpha_i=0$$

用二次规划方法求得α_s>0 s∈S（α_i=0 时，返回空向量）。用公式（**3**）表示支持向量 w：

$$w=\sum_{s\in S}\alpha_s y_s x^{(s)} \qquad (5)$$

α_s满足以下等式（结合式（**1**）和式（**2**））：

$$y_s = \left(w \cdot x^{(s)} + b\right) = 1$$

将 w 替换为式（**3**），两边同时乘以 y_s（+1 或-1），得到：

$$b = y_s - \sum_{m \in S} \alpha_m y_m x^{(m)} \cdot x^{(s)}$$

取所有支持向量 N_s 的均值，对参数 b 的估计更加准确：

$$b = \frac{1}{N} \sum_{s \in S} y_s - \sum_{m \in S} a_m y_m x^{(m)} \cdot x^{(s)} \qquad (6)$$

我们根据式（**5**）和式（**6**）得到了支持向量机算法的参数值。然后，就可以用这些参数值定义的支持向量机模型预测测试集数据点 t 的类别：

$$x^{(t)} \cdot w + b \geqslant 1 \to y_t = 1$$
$$x^{(t)} \cdot w + b \leqslant -1 \to y_t = -1$$

如果一条线无法恰好将所有数据点分到两个类别，我们需要允许数据点以多大的比率 $\xi_i > 0$ 被分错类，即

$$x^{(t)} \cdot w + b \geqslant 1 - \xi_i \to y_t = 1$$
$$x^{(t)} \cdot w + b \leqslant -1 + \xi_i \to y_t = -1$$

我们需要最大化间隔，以最小化将数据点分错类别的情况。该条件可以表述为：

$$min \frac{1}{2} |w|^2 + C \sum_{i=0}^{N-1} \xi_i \text{such that } y_i \left(w \cdot x^{(i)} + b\right) - 1 + \xi_i \geqslant 0 \forall i \in 1,..,N$$

其中，参数 C 用来平衡间隔大小跟分类错误之间的关系（C=0 表示没有分错类别的，间隔最大化；C>1，很多数据点分错类别，间隔较小）。使用跟之前相同的方法，可将上面这个问题转换为带有约束条件的对偶问题，注意拉格朗日乘子以 C 为上界：

$$\max_{\alpha_i} \left[\sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i H_{ij} \alpha_j \right], C \geqslant \alpha_i \geqslant 0 \forall i \in 0,..,N-1, \sum_{i=0}^{N-1} y_i \alpha_i = 0$$

到目前为止，我们处理的是二值分类问题。真实的分类问题，也许有多个类别，用支持向量机处理多个类别的分类问题，常用的方法（对数几率回归一节讲过）有两种：一对多或一对一。给定包含 M 个类别的分类问题：方法一，训练 M 个 SVM 模型，每个为目标类别 j 输出+1，其余类别输出-1。方法二，分别为每组类别 i 和 j 训练一个模型，共训练 $\frac{M(M-1)}{2}$ 个模型。显然，方法二计算开销更大，但结果也更为准确。

类似，SVM 也可用于 y_i 是介于-1 到 1 之间的回归问题。这时，算法的目标是寻找参数 w 和 b，使得

$$y_i = w \cdot x^{(i)} + b$$

我们假设实际值 t_i 可能不同于预测值 y_i 的最大误差为∈。在容忍误差∈的基础上，我们还能进一步容忍 ξ_i^+ 或 ξ_i^- 大小的误差，这具体取决于 y_i 是大于还是小于 t_i 。在图 3.4 中，数据点 i 的多个预测值 y_i 围绕在实际值 t_i 的周围，同时还给出误差范围。

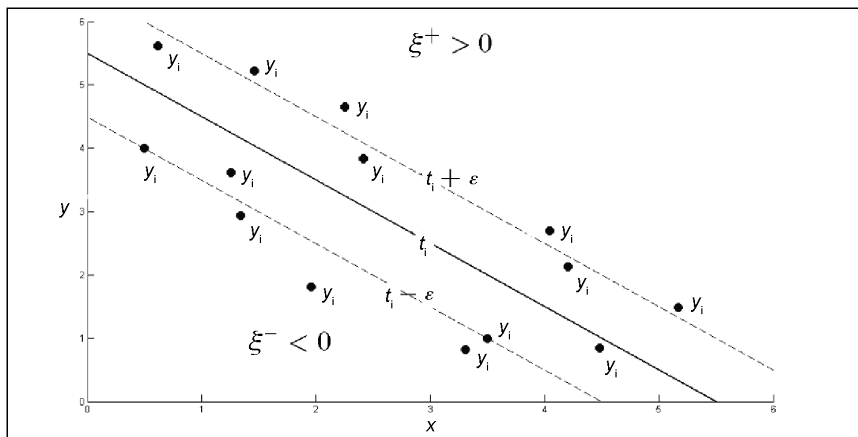


图 3.4 预测值 y_i 分布在实际值 t_i 的周围

最小化问题变为：

$$min\frac{1}{2}|w|^2 + C\sum_{i=0}^{N-1}\left(\zeta_i^+ + \zeta_i^-\right)$$

其中：

$$t_i - y_i \leqslant \in + \zeta_i^+, t_i - y_i \geqslant - \in - \zeta_i^-, \zeta_i^- > 0, \zeta_i^- > 0 \forall i \in 0,..,N-1$$

不难看出，相应的对偶问题变为：

$$\max_{\alpha_i^+,\alpha_i^-}\left[\sum_{i=0}^{N-1}\left(\alpha_i^+ - \alpha_i^-\right)t_i - \in\sum_{i=0}^{N-1}\left(\alpha_i^+ - \alpha_i^-\right) - \frac{1}{2}\sum_{i,j}\left(\alpha_i^+ - \alpha_i^-\right)\left(\alpha_j^+ - \alpha_j^+\right)x^i \cdot x^i\right]\text{subject}$$

$$\text{to } C \geqslant \alpha_i^+, \alpha_i^- \geqslant 0 \forall i \in 0,..,N-1, \sum_{i=0}^{N-1}\left(\alpha_i^+ - \alpha_i^-\right) = 0$$

其中， α_i^+ 、 α_i^- 为拉格朗日乘子。

新预测值 y_p 可用公式 $y_p = \sum_{i=0}^{N-1}\left(\alpha_i^+ - \alpha_i^-\right)x^i \cdot x^p + b$ 求得，参数 b 的计算方法跟之前相同——

选取 S 的子集中满足条件 $C>\alpha_s^+$、$\alpha_s^->0$ 和 ξ_s^+、$\xi_s^-=0$ 的支持向量求解，取均值①：

$$b=\frac{1}{N_s}\sum_{s\in S}t_s-\in-\sum_{m\in S}\left(\alpha_m^+-\alpha_m^-\right)y_m x^{(m)}\cdot x^{(s)}$$

内核技巧

有些数据集在特定的空间线性不可分，但转换到合适的空间后，可用超平面合理地将数据分成两个或多个类别。如图 3.5 所示的这个例子：

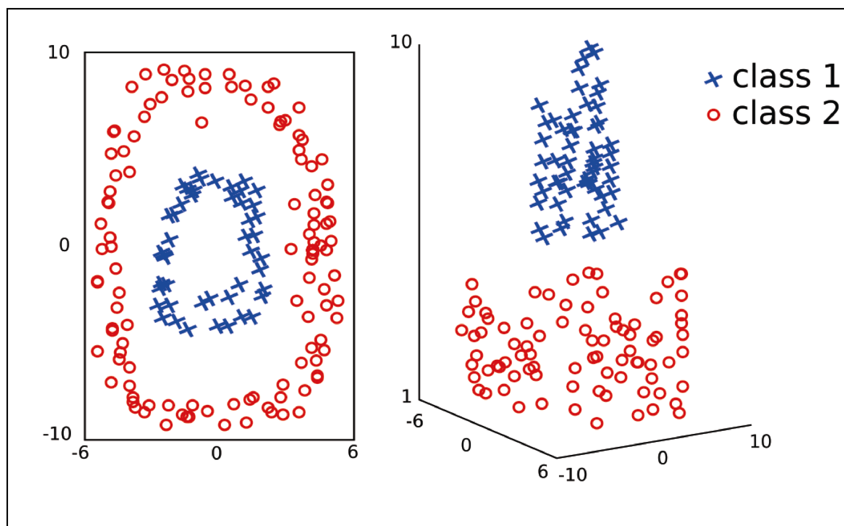


图 3.5　左侧的数据集在二维空间不可分，映射到三维空间后，两个类别变为可分

图 3.5，两个类别在二维空间（左图）显然是线性不可分的。假如我们对数据应用内核函数 K，使得：

$$K\left(x^{(i)},x^{(j)}\right)=e^{\frac{-\left|x^{(i)},x^{(j)}\right|^2}{2\sigma^2}}$$

现在就可以用二维平面（右图）将数据分开。SVM 算法的内核函数是作用于矩阵 H_{ij} 的，记得用内核函数替换变量 i、j 的点积：

$$H_{ij}=y_i y_j x^{(i)}\cdot x^{(j)}\rightarrow H_{ij}=y_i y_j K\left(x^{(i)},x^{(j)}\right)$$

① 原文为“averaging on the subset S given by the support vectors associated with the subset $C>\alpha_s^+,\alpha_s^->0$ and $\xi_i^+,\xi_i^-=0$”。——译者注

SVM 算法常用的内核函数有：

- 线性内核：$K\left(x^{(i)},x^{(j)}\right)=x^{(i)}\cdot x^{(j)}$
- 径向基（Radial Basis Kernel，RBF）内核：$K\left(x^{(i)},x^{(j)}\right)=e^{\frac{-\left|x^{(i)}-x^{(j)}\right|^2}{2\sigma^2}}$
- 多项式内核：$K\left(x^{(i)},x^{(j)}\right)=\left(x^{(i)}\cdot x^{(j)}+a\right)^b$
- Sigmoid 内核：$K\left(x^{(i)},x^{(j)}\right)=\tanh\left(ax^{(i)}\cdot x^{(j)}-b\right)$

3.6 有监督学习方法的对比

我们接下来解决一个回归问题和一个分类问题，以测试本章前面讨论的这几种方法。为了避免过拟合问题，数据集分为两部分：训练集，找到能够拟合模型的参数；测试集，评估模型的正确率。然而，也许还有必要使用第 3 个集合——验证集，优化超参数（hyperparameter，比如 SVM 的 C 和∈、岭回归的 α）。原始数据集也许太小，因此分成三份不合适。此外，算法的结果也许会受到训练集、验证集和测试集所选用的特定数据点的影响。这两个问题，常用的解决方法是，用交叉检验方法评估模型——将数据集分成 k 个子集（叫作折），并按如下步骤训练模型。

- 用 k-1 折数据作为训练数据训练模型。
- 用剩余数据测试得到的模型。
- 重复以上步骤 k 次（一开始确定的折数），每次用另外 k-1 折数据训练（因此每次使用的测试集也就不同）。对 k 次迭代得到的正确率取均值，得到最终的正确率。

3.6.1 回归问题

下面我们结合波士顿郊区房屋数据集评估各种回归算法的优劣，该数据集可从 http://archive.ics.uci.edu/ml/datasets/Housing 和作者 GitHub 主页本章的文件夹中下载（https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_3/）。这一部分的代码也可从该文件夹找到。房屋数据集共有 13 个特征。

- **CRIM**：城镇人均犯罪率；
- **ZN**：每 2.5 万平方英尺土地中，规划的住宅用地的比例；

- **INDUS**：每个城镇用于非零售业务土地的比例；
- **CHAS**：查尔斯河，哑变量（靠近河为 1；否则为 0）；
- **NOX**：氮氧化物的浓度（百万分之一）；
- **RM**：住所平均拥有的房间数；
- **AGE**：建于 1940 年之前的业主自住房比例；
- **DIS**：与波士顿 5 个就业中心距离的加权值；
- **RAD**：驶往放射状高速公路的便捷指数；
- **TAX**：每$10000 美元，全额地税税率；
- **PTRATIO**：城镇师生比例；
- **B**：1000(Bk - 0.63)^2，其中 Bk 为城镇黑人人口的比例；
- **LSTAT**：住房条件较差人口的比例，我们想预测的是房屋价值 MEDV（以$1000 计）。

为了评估模型的质量，我们计算模型的均方误差和判定系数 R^2（coefficient of determination）。R^2 的定义为：

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1}\left(y_i - y_i^{pred}\right)^2}{\sum_{i=0}^{N-1}\left(y_i - y\right)^2}, \ y = \frac{\sum_{i=0}^{N-1} y_i}{N}$$

其中，y_1^{pred} 表示模型给出的预测标签。

若模型完美地拟合数据，R^2=1，模型的预测效果达到最佳。而 R^2=0 时，模型为一条恒等的直线（如果为负数，那就表明拟合程度很糟糕）。下面代码，训练线性回归、岭回归、套索回归和 SVM 回归模型，并计算各自的均方误差和判定系数，我们用到了 sklearn 库（IPython notebook 版代码文件请见 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_3/）。

```
In [4]: import pandas as pd
        import numpy as np
        from sklearn import cross_validation
        from sklearn import svm
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.linear_model import LinearRegression
        from sklearn.linear_model import Ridge
        from sklearn.linear_model import Lasso
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn.metrics import mean_squared_error
```

```
In [7]: df = pd.read_csv('housing.csv',sep=',',header=None)
        #shuffle the data
        df = df.iloc[np.random.permutation(len(df))]
        X= df[df.columns[:-1]].values
        Y = df[df.columns[-1]].values

        cv = 10
        print 'linear regression'
        lin = LinearRegression()
        scores = cross_validation.cross_val_score(lin, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(lin, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'ridge regression'
        ridge = Ridge(alpha=1.0)
        scores = cross_validation.cross_val_score(ridge, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(ridge, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'lasso regression'
        lasso = Lasso(alpha=0.1)
        scores = cross_validation.cross_val_score(lasso, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(lasso, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'decision tree regression'
        tree = DecisionTreeRegressor(random_state=0)
        scores = cross_validation.cross_val_score(tree, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(tree, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'random forest regression'
        forest = RandomForestRegressor(n_estimators=50, max_depth=None,min_samples_split=1,
                                       random_state=0)
        scores = cross_validation.cross_val_score(forest, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(forest, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)
        #svm
        print 'linear support vector machine'
        svm_lin = svm.SVR(epsilon=0.2,kernel='linear',C=1)
        scores = cross_validation.cross_val_score(svm_lin, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(svm_lin, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'support vector machine rbf'
        clf = svm.SVR(epsilon=0.2,kernel='rbf',C=1.)
        scores = cross_validation.cross_val_score(clf, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'knn'
        knn = KNeighborsRegressor()
        scores = cross_validation.cross_val_score(knn, X, Y, cv=cv)
        print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(knn, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)
```

用 pandas 库加载住房数据，用 df.iloc[np.random.permutation(len(df))]打乱数据的顺序，从而在使用交叉检验方法时，保证各折数据是随机选取的（我们使用 10 折）。各算法的输出结果如下：

```
linear regression
mean R2: 0.72 (+/- 0.15)
MSE: 23.5515499366
ridge regression
mean R2: 0.72 (+/- 0.16)
MSE: 23.7397585761
lasso regression
mean R2: 0.71 (+/- 0.17)
MSE: 24.734860679
decision tree regression
mean R2: 0.75 (+/- 0.24)
MSE: 19.8023913043
random forest regression
mean R2: 0.87 (+/- 0.12)
MSE: 10.9910313913
linear support vector machine
mean R2: 0.70 (+/- 0.25)
MSE: 25.833801836
support vector machine rbf
mean R2: -0.01 (+/- 0.11)
MSE: 83.8283880541
knn
mean R2: 0.54 (+/- 0.23)
MSE: 37.8792632411
```

随机森林算法（50 棵树）训练的模型对数据的拟合程度最高；它返回的判定系数均值为 0.86，MSE=11.5。决策树回归符合预期，它的 R^2 比随机森林小，MSE 则比随机森林高（这两个指标分别为 0.67 和 25）。内核为 **rbf**（C=1，∈=0.2）的支持向量机是最差的模型，其 MSE 高达 83.9，R^2 则为 0.0。而线性内核 SVM 是一个非常出色的模型（R^2 和 MSE 分别为 0.69 和 25.8）。套索回归和岭回归，结果相差不大，R^2 和 MSE 分别约为 0.7 和 24。提升模型效果的一个重要方法就是特征选取。因为在所有特征中，往往只有一部分特征适用于模型训练，而其他特征也许对模型的 R^2 没有任何贡献。特征选取也许会改善 R^2，因为将起误导作用的数据排除在外，同时训练时间也会相应缩短（考虑的特征更少）。

为特定模型抽取最佳特征，方法有很多，我们将探索递归特征消除法（Recursive Feature Elimination，RSE），它只选取具有最大绝对权重的特征，直到特征的数量满足为止。SVM 算法，权重就是 w 的值，而回归算法，权重为模型的参数 θ。我们接下来利用 sklearn 的内置函数 RFE，仅用 4 个最佳特征（best_features）来观察模型的预测效果：

```
In [9]: from sklearn.feature_selection import RFE
        best_features=4
        print 'feature selection on linear regression'
        rfe_lin = RFE(lin,best_features).fit(X,Y)
        mask = np.array(rfe_lin.support_)
        scores = cross_validation.cross_val_score(lin, X[:,mask], Y, cv=cv)
        print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(lin, X[:,mask],Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'feature selection ridge regression'
        rfe_ridge = RFE(ridge,best_features).fit(X,Y)
        mask = np.array(rfe_ridge.support_)
        scores = cross_validation.cross_val_score(ridge, X[:,mask], Y, cv=cv)
        print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(ridge, X[:,mask],Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'feature selection on lasso regression'
        rfe_lasso = RFE(lasso,best_features).fit(X,Y)
        mask = np.array(rfe_lasso.support_)
        scores = cross_validation.cross_val_score(lasso, X[:,mask], Y, cv=cv)
        print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(lasso, X[:,mask],Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'feature selection on decision tree'
        rfe_tree = RFE(tree,best_features).fit(X,Y)
        mask = np.array(rfe_tree.support_)
        scores = cross_validation.cross_val_score(tree, X[:,mask], Y, cv=cv)
        print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(tree, X[:,mask],Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'feature selection on random forest'
        rfe_forest = RFE(forest,best_features).fit(X,Y)
        mask = np.array(rfe_forest.support_)
        scores = cross_validation.cross_val_score(forest, X[:,mask], Y, cv=cv)
        print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(forest, X[:,mask],Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)

        print 'feature selection on linear support vector machine'
        rfe_svm = RFE(svm_lin,best_features).fit(X,Y)
        mask = np.array(rfe_svm.support_)
        scores = cross_validation.cross_val_score(svm_lin, X[:,mask], Y, cv=cv)
        print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
        predicted = cross_validation.cross_val_predict(svm_lin, X,Y, cv=cv)
        print 'MSE:',mean_squared_error(Y,predicted)
```

输出为：

```
feature selection on linear regression
R2: 0.61 (+/- 0.31)
MSE: 33.182126206
feature selection ridge regression
R2: 0.61 (+/- 0.32)
MSE: 33.2543979822
feature selection on lasso regression
R2: 0.68 (+/- 0.20)
MSE: 27.4174043724
feature selection on decision tree
R2: 0.70 (+/- 0.35)
MSE: 24.1185968379
feature selection on random forest
R2: 0.84 (+/- 0.14)
MSE: 13.6755712332
feature selection on linear support vector machine
R2: 0.60 (+/- 0.33)
```

RFE 函数返回布尔值列表（用 RFE 函数的 support_属性得到该列表），表明选择了哪些特征（值为 true）和没有选择哪些特征（值为 false）。我们用选择的特征评估前面所讲的几个模型。

即使仅使用 4 个特征，最佳模型仍是由 50 棵树组成的随机森林算法，它的 R^2 只是稍微低于用全部特征训练得到的模型（0.82 和 0.86 的差距）。其余模型——套索回归、岭回归、决策树和线性 SVM 回归——它们的 R^2 与随机森林有较大的差距，但比起之前用全部特征训练得到的模型，结果依旧可观。请注意，由于 KNN 算法不支持为特征加权，因此无法使用 REF 方法抽取特征。

3.6.2　分类问题

我们用汽车质量（inaccurate、accurate、good 和 very good）评估数据集，训练本章学习的分类器。该数据集含有 6 个描述汽车主要特点的特征（购买时的价格、维护成本、车门数量、核载人数、后备箱大小和安全性）。该数据集可以从 http://archive.ics.uci.edu/ml/datasets/Car+Evaluation 或我的 GitHub 主页本章文件夹下载，代码也放到这里了（https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_3/）。我们用准确率（precision）、召回率（recall）和 f 值评估算法的正确性。给定一个只包含两个类别（正类和负类）的数据集，我们将正确标记为正类的数据点称为真正类（true positive，tp），误标记为正类的点称为假正类（false positive，fp），误标记为负类的点称为假负类（false negative，fn）。有了以上定义，我们就可以给出准确率[①]、召回率和 f 值的公式：

$$Precision = \frac{tp}{tp + fn}$$

$$Recall = \frac{tp}{tp + fn}$$

$$f - measure = 2\frac{(precision \cdot recall)}{(precision + recall)}$$

分类问题，对于类别 C 而言，准确率（1.0）指分到类别 C 的每个数据点实际属于类别 C（不关注 C 类数据点是否分错类），召回率为 1.0 则指 C 类数据点都分到 C 类（但是不关注其他数据点是否被误分到 C 类）。

多个类别分类问题，有多少种类别标签，这些度量指标就计算多少次。每次将一个类别当作正类，其余当作负类。然后对三种不同指标的计算结果分别取均值，以此来估计总准确率、召回率和 f 值。

① 原书准确率公式有误，实际为 $Precision=\frac{tp}{tp+fp}$。——译者注

为汽车数据集分类代码如下。首先，导入所有的库，并将数据导入为pandas的DataFrame对象。

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn import cross_validation
        from sklearn import svm
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.linear_model import LogisticRegression
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import f1_score
        from sklearn.metrics import precision_score
        from sklearn.metrics import recall_score
```

```
In [10]: #read data in
         df = pd.read_csv('data_cars.csv',header=None)
         for i in range(len(df.columns)):
             df[i] = df[i].astype('category')
         df.head()
```

Out[10]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

下面几个特征，特征值为类别型：

```
buying 0        v-high, high, med, low
maintenance 1   v-high, high, med, low
doors 2         2, 3, 4, 5-more
persons 3       2, 4, more
lug_boot 4      small, med, big
safety 5        low, med, high
car evaluation 6 unacc,acc,good,vgood
```

将这些特征值转换为分类算法里使用的数字：

```
In [14]: #map catgories to values
         map0 = dict( zip( df[0].cat.categories, range( len(df[0].cat.categories ))))
         #print map0
         map1 = dict( zip( df[1].cat.categories, range( len(df[1].cat.categories ))))
         map2 = dict( zip( df[2].cat.categories, range( len(df[2].cat.categories ))))
         map3 = dict( zip( df[3].cat.categories, range( len(df[3].cat.categories ))))
         map4 = dict( zip( df[4].cat.categories, range( len(df[4].cat.categories ))))
         map5 = dict( zip( df[5].cat.categories, range( len(df[5].cat.categories ))))
         map6 = dict( zip( df[6].cat.categories, range( len(df[6].cat.categories ))))

         cat_cols = df.select_dtypes(['category']).columns
         df[cat_cols] = df[cat_cols].apply(lambda x: x.cat.codes)

         df = df.iloc[np.random.permutation(len(df))]
         print df.head()

              0  1  2  3  4  5  6
         570  0  0  1  0  1  1  2
         951  2  3  3  0  0  1  2
         1633 1  1  0  1  1  2  0
         412  3  1  3  0  0  2  2
         156  3  0  1  2  1  1  2
```

因为我们需要计算和保存所有方法分类效果的度量指标，为此我们编写一个标准函数 CalcMeasures，将类别标签向量 Y 和特征向量 X 分开：

```
In [40]: df_f1 = pd.DataFrame(columns=['method']+sorted(map6, key=map6.get))
         df_precision = pd.DataFrame(columns=['method']+sorted(map6, key=map6.get))
         df_recall = pd.DataFrame(columns=['method']+sorted(map6, key=map6.get))
         def CalcMeasures(method,y_pred,y_true,df_f1=df_f1
                          ,df_precision=df_precision,df_recall=df_recall):

             df_f1.loc[len(df_f1)] = [method]+list(f1_score(y_pred,y_true,average=None))
             df_precision.loc[len(df_precision)] = [method]+list(precision_score(y_pred,y_true,average=None))
             df_recall.loc[len(df_recall)] = [method]+list(recall_score(y_pred,y_true,average=None))

         X= df[df.columns[:-1]].values
         Y = df[df.columns[-1]].values
```

我们采用 10 折交叉检验，代码如下：

```
In [41]: cv = 10
         method = 'linear support vector machine'
         clf = svm.SVC(kernel='linear',C=50)
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'rbf support vector machine'
         clf = svm.SVC(kernel='rbf',C=50)
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'poly support vector machine'
         clf = svm.SVC(kernel='poly',C=50)
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'decision tree'
         clf = DecisionTreeClassifier(random_state=0)
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'random forest'
         clf = RandomForestClassifier(n_estimators=50,random_state=0,max_features=None)
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'naive bayes'
         clf = MultinomialNB()
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'logistic regression'
         clf = LogisticRegression()
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)

         method = 'k nearest neighbours'
         clf = KNeighborsClassifier(weights='distance',n_neighbors=5)
         y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
         CalcMeasures(method,y_pred,Y)
```

我们将各种分类算法的 f 值、准确率和召回率这 3 个指标的度量结果存储到 DataFrame 对象中：

```
In [45]: df_f1
Out[45]:
```

	method	acc	good	unacc	vgood
0	linear support vector machine	0.271318	0.000000	0.846757	0.000000
1	rbf support vector machine	0.990921	1.000000	0.997933	0.984375
2	poly support vector machine	0.788918	0.841270	0.938010	0.800000
3	decision tree	0.957309	0.882353	0.989238	0.946565
4	random forest	0.963918	0.915493	0.991275	0.961832
5	naive bayes	0.040404	0.000000	0.825701	0.000000
6	logistic regression	0.265781	0.000000	0.820967	0.078947
7	k nearest neighbours	0.801609	0.534653	0.952988	0.666667

```
In [46]: df_precision
Out[46]:
```

	method	acc	good	unacc	vgood
0	linear support vector machine	0.182292	0.000000	0.981818	0.000000
1	rbf support vector machine	0.994792	1.000000	0.997521	0.969231
2	poly support vector machine	0.778646	0.768116	0.950413	0.738462
3	decision tree	0.963542	0.869565	0.987603	0.953846
4	random forest	0.973958	0.942029	0.985950	0.969231
5	naive bayes	0.020833	0.000000	0.998347	0.000000
6	logistic regression	0.208333	0.000000	0.919008	0.046154
7	k nearest neighbours	0.778646	0.391304	0.988430	0.507692

```
In [47]: df_recall
Out[47]:
```

	method	acc	good	unacc	vgood
0	linear support vector machine	0.530303	0.000000	0.744361	0.000000
1	rbf support vector machine	0.987080	1.000000	0.998346	1.000000
2	poly support vector machine	0.799465	0.929825	0.925926	0.872727
3	decision tree	0.951157	0.895522	0.990879	0.939394
4	random forest	0.954082	0.890411	0.996658	0.954545
5	naive bayes	0.666667	0.000000	0.703963	0.000000
6	logistic regression	0.366972	0.000000	0.741828	0.272727
7	k nearest neighbours	0.825967	0.843750	0.920000	0.970588

每个指标、每种分类算法，都要计算 4 次——“car evaluation”有多少个不同的类别

标签，就计算多少次。按照如下类别标签和索引的对照关系填充 DataFrame 各列的值。

```
'acc': 0, 'unacc': 2, 'good': 1, 'vgood': 3
```

从以上输出结果可见，最佳模型为 rbf 内核（C=50）SVM，但随机森林（50 棵树）和决策树的分类能力也很出色（对于以上 4 个类别，各项指标均在 0.9 之上）。朴素贝叶斯、对数几率回归和线性内核（C=50）SVM 训练的模型表现较差，尤其是对于 accurate、good 和 very good 3 个类别，准确率、召回率和 f 值 3 个指标均不理想，因为只有少数数据点打有以上类别标签：

```
In [42]: labels_counts=df[6].value_counts()
         pd.Series(map6).map(labels_counts)

Out[42]: acc       384
         good       69
         unacc    1210
         vgood      65
         dtype: int64
```

从百分比情况来看，very good（vgood）和 good 分别为 3.993% 和 3.762%，accurate 为 22.222%，而 inaccurate 为 70.0223% 。因此我们可以得出这样的结论：这些算法不适合预测数据集中的小众类别。

3.7 隐马尔可夫模型

虽然严格来讲，该方法不能算作有监督学习算法，但它也可以处理类似分类这样的问题，因此我们决定讲一讲该方法。为了帮助你更好理解，我们结合一个例子来讲。例如，推销员站在你面前，怎样通过观察他说话时的眼神（目光接触、向下看、向一侧看，观察结果 O_i 分别取 0、1 和 2），预测他是否在撒谎（两个状态 s_i $i \in 0,1$）。假如你观察到他的眼神序列 $O = O_{0'}, O_{1'}, O_{2'}, O_{3'}, O_{4'}, \cdots$，为 0, 1, 0, 2,⋯，我们想推断在连续的 t、t+1 时刻（推销员说前后两句话时），状态 S_i 的转移矩阵 A：

$$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.6 & 0.4 \end{bmatrix} a_i = \sum_j a_{ij} = 1,, a_{ij} = P\left(s_i \ at\ t+1 \middle| s_j \ at\ t\right)$$

- 矩阵 A 的每个元素 a_{ij} 表示给定 t 时刻的状态 S_i，t+1 时刻位于状态 S_j 的概率。因而，0.3(a_{01})表示推销员 t 时刻撒谎之后，t+1 时刻不再撒谎的概率，0.6(a_{10})表示先不撒谎、后再撒谎的概率，0.7(a_{00})表示他接连在 t、t+1 时刻撒谎的概率，0.4(a_{11})

表示他 t 时刻是真诚的，t+1 时刻仍不撒谎的概率。类似地，我们还可以定义矩阵 B[①]，将他是否撒谎两种状态和 3 种可能的行为联系起来：

$$B=\begin{bmatrix}0.7 & 0.1 & 0.2\\ 0.1 & 0.6 & 0.3\end{bmatrix} b(k)=\sum_j b_j(k)=1, b_j(k)=P(k\ at\ t|s_j\ at\ t)$$

B 中的元素 $b_{j(k)}$ 表示，给定 t 时刻的状态 s_j，t 时刻观察到 k 的概率。例如，0.7(b_{00})、0.1(b_{01})和 0.2(b_{02})分别表示，我们观察到推销员以下行为后，他撒谎的概率——目光接触、向下看和向一侧看。是否撒谎和 3 种行为之间的关系表述如图 3.6 所示：

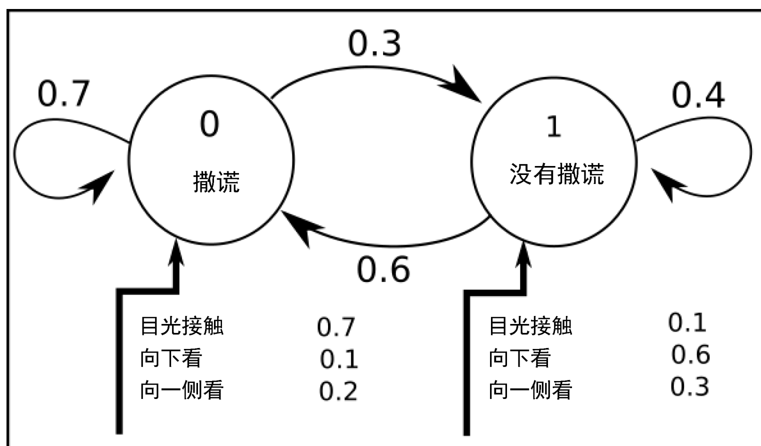


图 3.6 推销员行为——包含两个状态的隐马尔可夫模型

推销员的初始状态分布，可以定义为 π=[0.6,0.4]（0 时刻说第 1 句话时，他说谎的可能性稍大一些）。请注意，矩阵 π、A、B 为行随机矩阵（row stochastic），每行元素之和为 1，并且每行跟时间没有直接关系。**隐马尔可夫模型**（Hidden Markov Model，HMM）由 3 个矩阵组成（λ=(π, A, B)），描述的是观察序列 $O=O_0,O_1,\cdots,O_{T-1}$ 和相应的隐藏状态序列 $S=S_0,S_1,\cdots,S_{T-1}$ 之间的关系。该算法通常使用如下标记符号：

- T 为观察序列 $O=O_{0'},O_{1'},\cdots,O_{T-1}$、隐藏状态序列 $S=S_{0'},S_{1'},\cdots,S_{T-1}$ 的长度；
- N 为模型中可能（隐藏）的状态数量；
- M 为可能的观察结果数量：$O_k, k\in 0,1,\cdots, M\text{-}1$；
- A 为状态转换矩阵；

① 又称发射概率矩阵。B 的行表示状态，列表示行为。该例中，B 的第 0 行，状态为撒谎；第 1 行状态为不撒谎。第 0 列，目光接触；第 1 列，向下看；第 2 列向一侧看。——译者注

- B 为观察概率矩阵；
- π 为初始状态的概率分布。

推销员有没有撒谎这个例子，M=3、N=2。假如我们想根据自己观察到的推销员的行为序列 $O = O_{0'}, O_{1'}, \cdots, O_{T-1}$，预测他在讲话过程 $S_{0'}, S_{1'}, \cdots, S_{T-1}$ 有没有撒谎。我们可以计算每个状态序列 S 的概率，找出最可能的状态序列：

$$P(S) = \pi_{s_0} b_{s_0}(O_0) a_{s_0 s_1} b_{s_1}(O_1) \ldots a_{s_{T-2} s_{T-1}} b_{s_{T-1}}(O_{T-1})$$

例如，时刻 T=4、状态序列 S=0101 和观察序列 O=1012 时[①]：

$$p(0101) = 0.6(0.1)(0.3)(0.1)(0.6)(0.1)(0.3)(0.3) = 9.722 \times 10^{-6}$$

同理，我们可以计算其他所有隐藏状态组合的概率，从而找出最可能的序列 S。**Viterbi** 算法能够高效地找出最可能序列 S。它计算的是从时刻 0 到时刻 t（一直计算到 T-1 时刻）的局部序列（partial sequence）的最大概率。实际应用中，我们需要计算以下几个量：

- $\delta_0 = \pi_i b_i(O_0)$　$i \in 0, \cdots, N\text{-}1$
- 对于 t=1,⋯,T-1 和 i=0, ⋯,N-1，来自状态 j 的所有可能路径中，t 时刻在状态 i 的最大概率：$\delta_t(i) = \max_{j \in 1,..,N-1} \left[\delta_{T-1} a_{ji} b_i(O_t)\right]$。$\delta_t(i)$ 的最大值对应的局部序列是从时刻 0 到 t 最可能的局部序列。
- 最终最可能的序列为 T-1 时刻概率最大的序列：$\delta_{T-1}(i)$。

例如，给定上面这个模型，长度为 T=2 的、最可能的序列计算方法如下[②]：

- P(10)=0.0024
- P(00)=0.0294

° 因此 $\delta_1(0)$=P(00)=0.0294

- P(01)=0.076
- P(11)=0.01

° 因此 $\delta_1(1)$=P(01)=0.076

最终最可能的序列是 00（连续两句话都撒谎）。

---

① S=0101 表示先撒谎，后不撒谎，然后再撒谎，最后不撒谎。O=1012 表示先向下看，又目光接触，然后又向下看，最后向一侧看。计算 p(0101)的式子，8 个概率值依次是一上来就撒谎的概率，撒谎时向下看的概率、撒谎转移至不撒谎状态的概率、不撒谎时目光接触的概率、不撒谎转移至撒谎状态的概率、撒谎时向下看的概率、撒谎转移至不撒谎状态的概率、不撒谎时向一侧看的概率。——译者注

② P(10)、P(01)和 P(11)3 个结果疑似有误。笔者提交了勘误，尚在确认中。——译者注

另一种寻找最可能序列的方法是，最大化正确状态的数量；也就是说，在每个时刻 t，状态 i 的概率都要达到最大 $\max_i (Y_t(i))$ 。我们可以用向后算法（backward algorithm）计算给定状态 i 的概率 $Y_t(i)$：

$$Y_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

其中：

- $P(O|\lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i)$
- $\alpha_0(i) = \pi_i b_i(O_0) i \in 0,..,N-1$ 和 $\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\right] b_i(O_t)$

从初始时刻到时刻 t，HMM 在时刻 t 位于状态 i，局部观察序列的概率：$\alpha_t(i) = P(O_{0,.}.,O_{t,}s_t = i|\lambda)$

- $\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$，其中$t < T-1$ 且 $\beta_{T-1}(i) = 1 \ \ i \in 1,\cdots,N-1$

t 时刻状态为 i，t 时刻直到 T-1 时刻，局部序列的概率：$\beta_t(i) = P(O_{t+1},\cdots,O_{t-1},S_t = i|\lambda)$

- 结合 t 时刻前后位于状态 i 的概率，求得 $Y_t(i) = \frac{a_t(i)\beta_t(i)}{P(O|\lambda)}$

请注意，以上两种计算最可能序列的方法所得到的结果不一定相同。

寻找最佳序列的逆问题——给定序列 $O = O_{0'}, O_{1'}, \cdots, O_{T-1}$ 和参数值 N、M，寻找最佳的 HMM λ=(π, A, B)——可以用 **Baum-Welch 算法**迭代求解。t 时刻位于状态 i，t+1 时刻向状态 j 移动的概率定义为：

$$Y_t(i,j) = P(s_t = i, s_{t+1} = j|O,\lambda) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}$$

其中，$Y_t(i) = \sum_{j=0}^{N-1} Y_t(i,j) \ for \ T \in 0,\cdots,T-2$ 且 $Y_{T-1}(i) = \frac{\alpha_{T-1}}{P(O|\lambda)}$。

Baum-Welch 算法描述如下：

- 初始化 λ=(π, A, B)
- 计算 $\alpha_t(i)$ 、$\beta_t(i)$ 、$Y_t(i,j)$ 和 $Y_t(i)$ $i \in 0,\cdots, N$-1

- 重新计算模型中的各个矩阵：

$$\pi(i)=Y_0(i),a_{ij}=\frac{\sum_{t=0}^{T-2}Y_t(i,j)}{\sum_{t=0}^{T-2}Y_t(i,j)},b_j(O_k)=\frac{\sum_{t=0}^{T-1}\delta o_t o_k Y_t(j)}{\sum_{t=0}^{T-1}Y_t(j)}$$

其中，$i,j\in 0,\cdots,N-1$；$k\in 0,\cdots,M-1$；δ_{ij} 为内罗克符号（Kronacker symbol），当 $i=j$ 时，值为 1，否则为 0。

- 迭代直到收敛：$P(O|\lambda)=\sum_{t=0}^{N-1}a_{T-1}(i)$

下一节，我们讲解如何用 Python 代码实现这些公式，测试 HMM 算法的效果。

HMM 算法的 Python 实现

我们下面要讨论的 hmm_example.py 文件照旧可从 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_3/下载。

我们先定义一个类，传入 HMM 模型的 3 个矩阵。

```
class HMM:
    def __init__(self):
        self.pi = pi
        self.A = A
        self.B = B
```

Viterbi 算法和正确状态最大化用以下两个函数实现：

```
def ViterbiSequence(self,observations):
    deltas = [{}]
    seq = {}
    N = self.A.shape[0]
    states = [i for i in range(N)]
    T = len(observations)
    #initialization
    for s in states:
        deltas[0][s] = self.pi[s]*self.B[s,observations[0]]
        seq[s] = [s]
    #compute Viterbi
    for t in range(1,T):
        deltas.append({})
```

```
        newseq = {}
        for s in states:
            (delta,state) = max((deltas[t-1][s0]*self.A[s0,s]*self.B[
s,observations[t]],s0) for s0 in states)
            deltas[t][s] = delta
            newseq[s] = seq[state] + [s]
        seq = newseq

    (delta,state) = max((deltas[T-1][s],s) for s in states)
    return delta,' sequence: ', seq[state]

def maxProbSequence(self,observations):
    N = self.A.shape[0]
    states = [i for i in range(N)]
    T = len(observations)
    M = self.B.shape[1]
    # alpha_t(i) = P(O_1 O_2 ... O_t, q_t = S_i | hmm)
    # Initialize alpha
    alpha = np.zeros((N,T))
    c = np.zeros(T) #scale factors
    alpha[:,0] = pi.T * self.B[:,observations[0]]
    c[0] = 1.0/np.sum(alpha[:,0])
    alpha[:,0] = c[0] * alpha[:,0]
    # Update alpha for each observation step
    for t in range(1,T):
        alpha[:,t] = np.dot(alpha[:,t-1].T, self.A).T *
self.B[:,observations[t]]
        c[t] = 1.0/np.sum(alpha[:,t])
        alpha[:,t] = c[t] * alpha[:,t]
    # beta_t(i) = P(O_t+1 O_t+2 ... O_T | q_t = S_i , hmm)
    # Initialize beta
    beta = np.zeros((N,T))
    beta[:,T-1] = 1
    beta[:,T-1] = c[T-1] * beta[:,T-1]
    # Update beta backwards froT end of sequence
    for t in range(len(observations)-1,0,-1):
        beta[:,t-1] = np.dot(self.A, (self.B[:,observations[t]] *
beta[:,t]))
        beta[:,t-1] = c[t-1] * beta[:,t-1]

    norm = np.sum(alpha[:,T-1])
    seq = ''
    for t in range(T):
```

```
        g,state = max(((beta[i,t]*alpha[i,t])/norm,i) for i in
states)
        seq +=str(state)

    return seq
```

由于概率的乘法运算可能导致数值下溢问题，所有的$\alpha_t(i)$、$\beta_t(i)$ $i\in 0,\cdots,N\text{-}1$都乘以了一个常数：

- $c_0 = \dfrac{1}{\sum_{j=0}^{N-1}\alpha_0(j)}$
- $c_t = \dfrac{1}{\sum_{j=0}^{N-1}\alpha'_{t-1}(j)a_{ji}b_i(O_t)}\alpha'_t(i) = \sum_{j=0}^{N-1}c_{t-1}\alpha'_{t-1}a_{ji}b_i(O_t)$，其中$\alpha'_0(i)=\alpha_0(i)$

现在，我们可以用推销员是否撒谎那个例子中的矩阵初始化 HMM 模型，然后用前面定义的两个函数计算最可能的序列：

```
pi = np.array([0.6, 0.4])
A = np.array([[0.7, 0.3],
                           [0.6, 0.4]])
B = np.array([[0.7, 0.1, 0.2],
                           [0.1, 0.6, 0.3]])
hmmguess = HMM(pi,A,B)
print 'Viterbi sequence:',hmmguess.ViterbiSequence(np.array([0,1,0,2]))
print 'max prob sequence:',hmmguess.maxProbSequence(np.array([0,1,0,2]))
```

结果为：

```
Viterbi: (0.0044, 'sequence: ', [0, 1, 0, 0])
Max prob sequence: 0100
```

上面这种特殊情况，两种方法返回相同的序列。修改初始化时使用的矩阵，两种算法也许会给出不同的结果。我们得到的行为序列（目光接触、向下看、目光接触、向一侧看）可能对应推销员状态序列（撒谎、没有撒谎、撒谎、撒谎）的概率为 0.0044。

给定观察序列和参数 N、M 之后，我们也可以实现 Baum-Welch 算法，寻找最佳的 HMM 模型。代码如下：

```
def train(self,observations,criterion):
```

```
    N = self.A.shape[0]
    T = len(observations)
    M = self.B.shape[1]

    A = self.A
    B = self.B
    pi = copy(self.pi)

    convergence = False
    while not convergence:

        # alpha_t(i) = P(O_1 O_2 ... O_t, q_t = S_i | hmm)
        # Initialize alpha
        alpha = np.zeros((N,T))
        c = np.zeros(T) #scale factors
        alpha[:,0] = pi.T * self.B[:,observations[0]]
        c[0] = 1.0/np.sum(alpha[:,0])
        alpha[:,0] = c[0] * alpha[:,0]
        # Update alpha for each observation step
        for t in range(1,T):
            alpha[:,t] = np.dot(alpha[:,t-1].T, self.A).T *
self.B[:,observations[t]]
            c[t] = 1.0/np.sum(alpha[:,t])
            alpha[:,t] = c[t] * alpha[:,t]

        #P(O=O_0,O_1,...,O_T-1 | hmm)
        P_O = np.sum(alpha[:,T-1])
        # beta_t(i) = P(O_t+1 O_t+2 ... O_T | q_t = S_i , hmm)
        # Initialize beta
        beta = np.zeros((N,T))
        beta[:,T-1] = 1
        beta[:,T-1] = c[T-1] * beta[:,T-1]
        # Update beta backwards froT end of sequence
        for t in range(len(observations)-1,0,-1):
            beta[:,t-1] = np.dot(self.A, (self.B[:,observations[t]] *
beta[:,t]))
            beta[:,t-1] = c[t-1] * beta[:,t-1]

        gi = np.zeros((N,N,T-1));

        for t in range(T-1):
            for i in range(N):

                gamma_num = alpha[i,t] * self.A[i,:] *
```

```
self.B[:,observations[t+1]].T * \
                            beta[:,t+1].T
                     gi[i,:,t] = gamma_num / P_O

        # gamma_t(i) = P(q_t = S_i | O, hmm)
        gamma = np.squeeze(np.sum(gi,axis=1))
        # Need final gamma element for new B
        prod = (alpha[:,T-1] * beta[:,T-1]).reshape((-1,1))
        gamma_T = prod/P_O
        gamma = np.hstack((gamma, gamma_T)) #append one Tore to
gamma!!!

        newpi = gamma[:,0]
        newA = np.sum(gi,2) / np.sum(gamma[:,:-1],axis=1).
reshape((-1,1))
        newB = copy(B)

        sumgamma = np.sum(gamma,axis=1)
        for ob_k in range(M):
            list_k = observations == ob_k
            newB[:,ob_k] = np.sum(gamma[:,list_k],axis=1) / sumgamma

        if np.max(abs(pi - newpi)) < criterion and \
               np.max(abs(A - newA)) < criterion and \
               np.max(abs(B - newB)) < criterion:
            convergence = True;

        A[:],B[:],pi[:] = newA,newB,newpi

    self.A[:] = newA
    self.B[:] = newB
    self.pi[:] = newpi
    self.gamma = gamma
```

请注意，上面代码用到了 copy 模块的浅复制方法，因此新创建的容器中装载的是对原始对象（pi、B）内容的引用。newpi 和 pi 是两个不同的对象，但是 newpi[0]是 pi[0]的引用。我们还用 NumPy 的 squeeze 函数，剔除了矩阵冗余的维度。

观察序列 O=0,1,0,2，对应的最佳模型是：

$$\pi = [1.00.0], A = \begin{bmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \end{bmatrix}, B = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.38 & 0.62 \end{bmatrix}$$

这表明，状态序列必须以推销员说真话开始，然后他持续在撒谎和不撒谎两种状态

间摇摆。推销员讲真话（或不撒谎），一定是目光接触，而撒谎时，要么向下看，要么向一侧看。

这个简单的 HMM 入门示例，我们假定每个观察值都是标量，但真实应用中，每个 O_i 通常是由多个特征组成的向量。HMM 算法常用来分类，有多少个类别，就训练多少个 HMM 模型 l_i，预测结果取概率最高的 $P(O|\lambda_i)$。接着这个例子往下想，假如我们要构造一台测谎机，检验每一位推销员。假定推销员所说的每句话（观察结果）O_i，我们都可以抽取：3 个视线特征，每个特征 e_i 有 3 个可能的取值（目光接触、向下看或向一侧看）；3 个声音特征，每个特征 v_i 有 3 个可能的取值（太大、太小或适中）；3 个手部特征，每个特征 h_i 有两个可能的取值（晃动或静止）。因此，观察结果序列 O_i 可表示为 $O_i=(e_i,v_i,h_i)$。训练阶段，我们请朋友撒谎。我们使用 Baum-Welch 算法，利用观察结果训练 HMM λ_0。然后，我们重复训练过程，再用真话训练 λ_1。预测阶段，我们记录推销员所说的话 O，计算 $P(O|\lambda_0)$ 和 $P(O|\lambda_1)$，取概率最大的作为预测结果。

虽然 HMM 算法已用于多个领域，但它在语音识别、手写字母识别和行为识别领域表现最好。

3.8 小结

在本章中，我们讨论了主要的分类和回归算法及其实现方法。你应该已经理解每种方法的使用场景，并懂得如何用 Python 语言和相关库（sklearn 和 pandas）实现这些方法。

下一章，我们将介绍最常用的 Web 数据挖掘技术，掌握从 Web 数据中学习的方法。

第 4 章 Web 挖掘技术

Web 数据挖掘技术适用于探索因特网上的数据，从中抽取相关信息。搜索网上内容，其过程很复杂，要用到多种算法，本章重点讲解这些算法。搜索引擎，拿到查询词（search query）之后，分析每个网页的数据，找到与查询词相关的网页。网页中的数据通常分为网页内容和链接到其他网页的超链接。一般而言，搜索引擎由以下部件组成：

- 采集网页的 Web 爬虫或蜘蛛；
- 抽取内容和预处理网页的解析器；
- 将网页组织为数据结构的索引器；
- 信息检索系统：根据文档与查询词的相关程度，找出最重要的文档；
- 以某种有意义的方式，调整各网页顺序的排序算法。

这些部件的核心技术为 Web 结构挖掘和 Web 内容挖掘。

搜索引擎的 Web 爬虫、索引器和排序机制，处理的是 Web 的结构（超链接文本形成的网络）。搜索引擎的其余部分（解析器和检索系统）为 Web 内容分析方法，因为要解析网页，检索其中的文本信息。

更一步来讲，对于收集到的网页，我们可以利用自然语言处理技术深入分析其中的内容，比如使用**潜在狄利克雷分布分析**（Latent Dirichlet Allocation，LDA）、意见挖掘或情感分析工具。这些重要技术适用于从 Web 内容抽取其发表人的主观看法。因此在很多市场营销、咨询领域的商业应用中，都能看到它们的身影。本章最后将讨论这些情感分析技术。现在，我们首先来讨论 Web 结构挖掘。

4.1 Web 结构挖掘

这一类 Web 挖掘技术，有两个主要任务，一是如何发现网页之间的关系，二是如何利用链接结构找出相关网页。任务一，我们通常用爬虫爬取链接，并将爬取到的链接和网页存储到索引器。任务二，则要计算网页的重要性，并按其排序。

4.1.1 Web 爬虫

爬虫从一组 URL（种子网页）开始爬取，从这些网页抽取链接后，接着去爬取它们。然后，再从新爬取到的网页抽取新链接。重复这一过程，直到满足预先设定的标准为止。未爬取的 URL 存储在 **frontier** 列表。根据爬虫怎样使用这个列表，我们可将爬虫算法分为不同的类型，比如广度优先（breadth-first）和有倾向性（preferential）的爬虫。广度优先算法，下一个要爬取的 URL 来自于 frontier 列表的头部位置，而新爬取的 URL 则追加到 frontier 列表的尾部。有倾向性的爬虫，则用特定的重要性评估方法，评估未访问的 URL 列表，以决定先爬取哪个页面。从网页抽取链接需要用解析器。Web 内容挖掘一节会做详细介绍。

爬虫本质上是一种图搜索算法，即按照特定规则，检索初始页各相邻页面的结构，规则可以是爬取的最大链接数量（图的深度）、爬取的最大网页数量或时间限制等。然后，爬虫从我们感兴趣的网页（比如信息港 hub 和权威网页 authority）抽取一部分内容。信息港指包含大量链接的网页，而权威网页指该网页的 URL 多次出现在其他网页（该指标可衡量网页的受欢迎程度）。我们后面会用 Scrapy 爬取页面。Scrapy 爬虫库很常用，它是用 Python 实现的，采用并发机制（用 Twisted 框架实现异步编程）加快处理速度。Scrapy 的使用教程见第 7 章电影推荐系统 Web 应用，我们要用它从影评抽取信息。

4.1.2 索引器

索引器是一种网页存储方式，它将爬虫爬取到的网页存储到结构化数据库，便于后续根据给定的查询词快速检索。最简单的索引方法是直接存储所有网页，查询时，查找包含关键词（查询词的组成部分）的所有文档。然而，如果网页很多（实际应用中确实如此），由于计算开销很大，这样做不可行。最常用的提升检索速度的方法叫作**倒排索引机制**（inverted index scheme），它为大多数主流搜索引擎所采用。

给定一组网页 $p_1, \cdots, p_k$ 和包含网页出现的所有单词 $w_i \in V$ 的词汇表 V，我们将单词及含有该单词的网页编号对应起来，形成如下形式的列表：w_i:$id_{p1}, id_{p3}, \cdots, w_i$: id_{p2}, id_{p7}，再将

这样的列表保存起来，做成倒排索引数据库。

其中，id_{p_j} 表示网页 j 的编号。除此之外，我们还可以存储每个单词的其他信息，比如单词在每个网页的频数或在网页中的位置。为了保证本书内容的完整性，我们介绍了这些主要概念。但索引器的具体实现，超出了本书的讲解范围。

因而，查询词若由多个单词组成，发起检索后，系统将查找每个单词的倒排索引列表，并将找到的多个倒排索引列表合并后返回。然后，再用排序算法和信息检索系统，综合评定文档和查询词之间的相关性，从而最终确定索引列表的顺序。

4.1.3 排序——PageRank 算法

排序算法很重要。因为通常仅一条查询词，也许就会返回海量网页。因此，如何选取最相关的网页成为一个大难题。进一步讲，欺骗信息检索模型也很简单，只需在网页中插入多个关键词，使得网页与大量查询词相关即可。因此，为了解决网页的重要性（排序分数）的评估问题，人们考虑利用互联网具有图结构这一特点。超链接图状结构——一个网页链接到另一个网页——是评估网页相关性的主要信息源。超链接可分为以下两种：

- 网页 i 的入链：指向网页 i 的超链接；
- 网页 i 的出链：网页 i 中指向其他网页的超链接。

直觉告诉我们，入链越多的网页应该越重要。超链接结构的研究是社交网络分析的一部分，研究人员研制了很多算法并已投入使用。但有的算法由于时间较长，如今已不为人所知。我们将解释最著名的 **PageRank** 算法。该算法是由 Sergey Brin 和 Larry Page（谷歌创始人）在 1998 年提出的。它的总体思想是：一个网页的权威性为指向它的所有网页的权威性之和。如果网页 j 的权威性为 $P(j)$，那么 j 指向的所有网页均分它的权威性，因此 j 的每个出链所得到的权威性等于 $P(j)/N_j$。用正式的数学语言来讲，网页 i 的权威性或 PageRank 值为：

$$P(i)=\sum_j A_{ji}P(j)$$

如果网页 j 指向网页 i，上式中的 $Aij=\frac{1}{N_1}$；否则，它为 0。A_{ji} 叫作邻接矩阵（adjacency matrix），它表示的是节点 j 输送给节点 i 的权威性。若图中共有 N 个节点，将各个变量换作相应的矩阵，上述公式可改写为如下形式：

$$P=A^T P, P=(P(1),\cdots,P(N))^T$$

如果邻接矩阵满足特定条件，上式等价于特征值 $\lambda=1$ 的特征系统。该公式还可以用马尔可夫链的相关术语来解释——A_{ij} 称为节点 j 到节点 i 的转移概率，节点 i 的权威性 $p(i)$则称为访问节点 i 的概率。两个（或更多）节点也许指向彼此，但不再指向其他节点。那么，访问过这两个节点之后，产生死循环，用户困于这两个节点之中。这种情况叫作 **rank sink**（矩阵 A 叫作**周期矩阵**），解决方法是增加逃脱因子 E，允许随机从每个网页跳到另一网页，而不用遵循矩阵 A 所刻画的马尔可夫链：

$$P=\left(\frac{(1-d)E}{N}+dA^T\right)P$$

其中，$E=ee^T$ 为 $N\times N$ 维的全 1 矩阵（e 为单位向量），d（damping factor，阻尼系数）表示转移矩阵 A 所定义的转移概率。（1-d）为随机访问一个网页的概率。公式表明，所有的节点相互连接，即使邻接矩阵某一行有多个元素为 0，比如节点 s 所在的行。那么，图状结构中，N 个访问 s 的节点，每个节点都以很小的概率 $1/N$ 访问 s，即 $A_{sj}=1/N$。A 为行随机矩阵，每行所有元素之和为 1；$\Sigma_j A_{ij}=1 \quad i\in 1,\cdots,N$（每行至少一个元素不为 0 或至少每个网页有一条出链）。我们对 P 向量进行规范化，使得 $e^Tp=N$，可将公式简化为：

$$P=(1-d)e+dA^TP\rightarrow P(i)=(1-d)+d\sum_{j=1}^{N}A_{ji}P(j)\forall i1,\cdots,\ N$$

上述公式可用幂迭代方法求解。第 8 章将用该算法实现影评情感分析系统。该算法的主要优点是，不依赖于查询词（因此 PageRank 值可离线计算好，查询时直接检索即可）。此外，它还具有较强的鲁棒性，能有效防范作弊，因为作弊者将有影响力的网页作为自己网页的入链不太现实。

4.2 Web 内容挖掘

这一类挖掘技术，着力解决如何从网页内容抽取信息。内容挖掘的常见过程描述如下：收集网页，整理网页内容（句法解析技术）；处理网页内容，删除文本中不重要的部分（自然语言处理技术）；然后，再用信息检索系统，为给定查询词匹配相关文档。下面我们将分别讨论这 3 种技术。

句法解析

网页为 HTML 格式，因此首先需要将相关信息从 HTML 代码中抽取出来。HTML 解析器构建标签树，便于从中抽取内容。现如今解析器有很多，我们拿 Scrapy 库当例子，它

提供了命令行解析器，详见第 7 章。比如，我们想解析维基百科的首页 https://en.wikipedia.org/wiki/Main_Page。只需在终端输入：

```
scrapy shell 'https://en.wikipedia.org/wiki/Main_Page'
```

然后，我们就可以在命令提示符后，用 response 对象和 xpath 语言解析网页。比如，获取网页的标题：

```
In [1]: response.xpath('//title/text()').extract()
Out[1]: [u'Wikipedia, the free encyclopedia']
```

或者，我们想抽取网页的所有链接（爬虫执行爬取工作前，我们需要指定链接，因此我们可以先用 Scrapy 库抽取链接）。链接通常用<a>标签，且 URL 为 href 的属性值：

```
In [2]: response.xpath("//a/@href").extract()
Out[2]:
[u'#mw-head',
 u'#p-search',
 u'/wiki/Wikipedia',
 u'/wiki/Free_content',
 u'/wiki/Encyclopedia',
 u'/wiki/Wikipedia:Introduction',
…
 u'//wikimediafoundation.org/',
 u'//www.mediawiki.org/']
```

我们可以考虑用鲁棒性更强的方法解析网页内容，因为网页的编写者通常不是程序员，所以 HTML 代码也许包含句法错误，并且浏览器可能根据自己的理解对其进行过修正。此外，网页也许包含大量广告信息，增加了解析相关信息的复杂度。如何识别网页的主要内容，人们提出了几种不同的算法（比如，树匹配），但写作本书时，还没有相关的 Python 库，因此我们不打算进一步讨论这个主题。然而，请注意 newspaper 库实现的解析功能非常好用，能够抽取网页文章的主体部分，第 7 章会用到该库。

4.3　自然语言处理

从网页抽取文本内容之后，通常要对文本数据做预处理，删除不提供任何相关信息的部分。对文本本进行分词操作，将文本转换为单词列表，删除所有标点符号。通常还要删除所有的停用词（stop words），这些词构成句子的句法，但不包含文本信息（例如，连词、

冠词和介词），比如 *a*、*about*、*an*、*are*、*as*、*at*、*be*、*by*、*for*、*from*、*how*、*in*、*is*、*of*、*on*、*or*、*that*、*the*、*these*、*this*、*to*、*was*、*what*、*when*、*where*、*who*、*will*、*with* 等。

很多英语（或其他语言）单词，词干相同，但前后缀不同。例如，*think*、*thinking* 和 *thinker* 词干都是 *think*，这表明它们的意思相同，但在句子中所起作用不同（动词、名词等）。将上面这样一组单词还原为词干的过程叫作**词干提取**（stemming），现如今，词干提取算法有多种（Porter、Snowball 和 Lancaster）。所有这些技术都属于**自然语言处理**这一更为广阔的算法范畴，nltk 库用 Python 语言实现了这些技术（照旧可用 sudo pip install nltk 安装该库）。举个例子，下面代码用前面提到的技术，处理一段示例文本（在终端启用 Python shell）：

```
>>> import nltk
>>> from nltk.tokenize import WordPunctTokenizer
>>> nltk.download('stopwords')
[nltk_data] Downloading package 'stopwords' to
[nltk_data]     /Users/andrea/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
>>> from nltk.corpus import stopwords
>>> stopwords = stopwords.words('english')
>>> tknzr = WordPunctTokenizer()
>>> from nltk.stem.porter import PorterStemmer
>>> stemmer = PorterStemmer()
>>> text = 'The European languages are members of the same family. Many words in a language trans
late into familiar words in another. For science, music, sport, etc, Europe uses the same vocabul
ary.  Everyone realizes why a new common language would be desirable: one could refuse to pay tra
nslators.'
>>> words = tknzr.tokenize(text)
>>> words
['The', 'European', 'languages', 'are', 'members', 'of', 'the', 'same', 'family', '.', 'Many', 'w
ords', 'in', 'a', 'language', 'translate', 'into', 'familiar', 'words', 'in', 'another', '.', 'Fo
r', 'science', ',', 'music', ',', 'sport', ',', 'etc', ',', 'Europe', 'uses', 'the', 'same', 'voc
abulary', '.', 'Everyone', 'realizes', 'why', 'a', 'new', 'common', 'language', 'would', 'be', 'd
esirable', ':', 'one', 'could', 'refuse', 'to', 'pay', 'translators', '.']
>>> words_clean = [w.lower() for w in words if w not in stopwords]
>>> words_clean
['the', 'european', 'languages', 'members', 'family', '.', 'many', 'words', 'language', 'translat
e', 'familiar', 'words', 'another', '.', 'for', 'science', ',', 'music', ',', 'sport', ',', 'etc'
, ',', 'europe', 'uses', 'vocabulary', '.', 'everyone', 'realizes', 'new', 'common', 'language',
'would', 'desirable', ':', 'one', 'could', 'refuse', 'pay', 'translators', '.']
>>> words_clean_stem = [stemmer.stem(w) for w in words_clean]
>>> words_clean_stem
['the', 'european', 'languag', 'member', 'famili', '.', 'mani', 'word', 'languag', 'translat', 'f
amiliar', 'word', 'anoth', '.', 'for', 'scienc', ',', 'music', ',', 'sport', ',', 'etc', ',', 'eu
rop', 'use', 'vocabulari', '.', 'everyon', 'realiz', 'new', 'common', 'languag', 'would', 'desir'
, ':', 'one', 'could', 'refus', 'pay', 'translat', '.']
```

注意，stopwords 列表是用 nltk dowloader 下载的，具体命令是 nltk.download('stopwords')。

4.3.1 信息检索模型

为给定查询词寻找最相关的文档，要用信息检索方法。为网页中的单词建模有多种方法，比如布尔模型、向量空间模型和概率模型。我们讲一下向量空间模型及其实现方法。我们用正式的语言来描述信息检索问题：给定由 V 个单词组成的词汇表和包含 N 个网页的

文档集，每个网页（或文档）d_i 可以看作是由单词组成的向量 $d_i=w_{1i},\ldots,w_{|V|i}$。$w_{ij}$ 表示每个单词 j 属于文档 i。根据选用的算法，w_{ij} 可以是数字（权重）或向量形式：

- **词频-逆文档频率**（Term Frequency-Inverse Document Frequency，TF-IDF），w_{ij} 是一个实数；
- **潜在语义分析**（Latent Semantic Analysis，LSA），w_{ij} 是一个实数（词的表示独立于文档 i）；
- **Doc2Vec**（或 word2vec），w_{ij} 由实数组成的向量（独立于文档 i 的表示）。

由于查询词也可以表示为单词向量 $q=w_{1q},\ldots,\ w_{|V|q}$，计算查询词向量和文档向量的相似度，可找到跟向量 q 最相似的网页。最常用的相似度度量方法叫作余弦相似度，对于任意文档 i，文档和查询词的相似度为：

$$\frac{d_i \bullet q}{|d_i||q|}=\frac{\sum_{j=1}^{|V|} w_{ij} w_{jq}}{\sqrt{\sum_{j=1}^{|V|} w_{ij}^2}\sqrt{\sum_{j=1}^{|V|} w_{qj}^2}}$$

注意，文献中还会使用其他度量方法（Okapi 和 pivoted normalization weighting），但是我们这里不多讲。

下面几节，我们详细介绍这 3 种建模方法，最后讲如何将其用于文本处理。

1．TF-IDF

该方法在计算 w_{ij} 时，考虑到了这样一个事实，在一个网页中出现多次并在大量网页出现的单词，其重要性低于在一个网页出现多次但仅在少数网页出现的单词。TF-IDF 值由以下两个因子相乘得到：

$w_{ij}=tf_{ij}xidf_i$，其中：

- $tf_{ij}=\frac{f_{ij}}{max\ f_{1i},\ldots,f_{|V|i}}$ 为文档 i 中单词 j 的归一化词频（normalized frequency）。
- $idf_i=\log\frac{N}{df_j}$ 为逆文档频率，df_j 为包含单词 j 的网页数量。

2．潜在语义分析（LSA）

该算法的名称来自于如下想法，每个单词（文档）在潜在空间可以得到充分地描述，

假定意思相似的单词在文本中出现的位置也相似。单词在子空间的映射，用（截断）奇异值分解方法来计算，该方法在第 2 章已介绍过。将奇异值分解方法用于 LSA 算法，那么包含所有网页的文档集用矩阵 X（$V×N$ 维）表示，其中每一列表示一篇文档：

$$X = U_t \sum_t V_t^T$$

U_t $(V'd)$为映射到 d 维新潜在空间的单词矩阵，$\Sigma_t V_t^T$ $(d'N)$为转换到子空间的文档的转置矩阵，Σ_t $(d'd)$为包含奇异值的对角矩阵。查询词向量按如下方式映射到潜在空间：

$$q_t = qU_t\Sigma_t^{-1}$$

接着，计算 V_t 的每一行所表示的文档跟 q_t 的余弦相似度。注意，文档在潜在空间的数学表达式为$\Sigma_t V_t$（不是 V_t），因为奇异值是空间轴的比例因子（scaling factor），因此需要将其考虑在内。因而，文档矩阵应该与$\Sigma_t q_t$ 相比较。然而，我们通常计算的是 V_t 和 q_t 的相似度，并且尚不清楚在实际应用中哪种方法效果最好。

3．Doc2Vec(word2vec)

该方法将每个单词 w_j 表示成独立于包含它的文档 d_i 的向量 v_{wj}。Doc2Vec 是 Mikolov 等人发明的 word2vec 算法的扩展，Doc2Vec 算法采用神经网络和反向传播方法生成词（和文档）向量。鉴于神经网络（尤其是深度学习）在很多机器学习应用中重要性日增，我们决定讲一下这种非常高级的方法，介绍其主要概念和公式。对于各个领域的机器学习，神经网络这种方法将变得极其重要。下面的内容是基于 Rong（2014）、Le 和 Mikolv（2014）的论文，下面提到的这些术语也正是当今文献所使用的。

4．word2vec——连续词袋和 Skip-gram 架构

将词汇表 V 中的每个单词 j 表示为长度为$|V|$的向量，向量的每个元素[①]为二进制 $x_j=(x_{1j}\cdots，x_{Vj})$，其中只有 $x_{jj}=1$，其余均为 0。word2vec 方法从两种网络架构（见图 4.1）中选取一种，训练由 N 个神经元（带权重）组成的一层（隐含层）神经网络。这两种架构都只有包含 N 个神经元或权重 h 的一层神经网络。因此，我们可将该方法看作是浅层学习而不是深度学习，后者通常指含有多层隐含层的网络。**连续词袋**（Continuous Bag Of Words，CBOW）方法（图 4.1 右侧）使用 C 个单词作为输入（叫作**上下文**）训练模型，尝试预测与输入的上下文相邻的单词（目标）。该方法的逆操作叫作 **Skip-gram**，它以目标单词作为输入，训练网络预测目标单词的上下文（图 4.1 左侧）。C 叫作窗口参数，它指定从距离目标词多远的位置选择上下文：

① 分量。——译者注

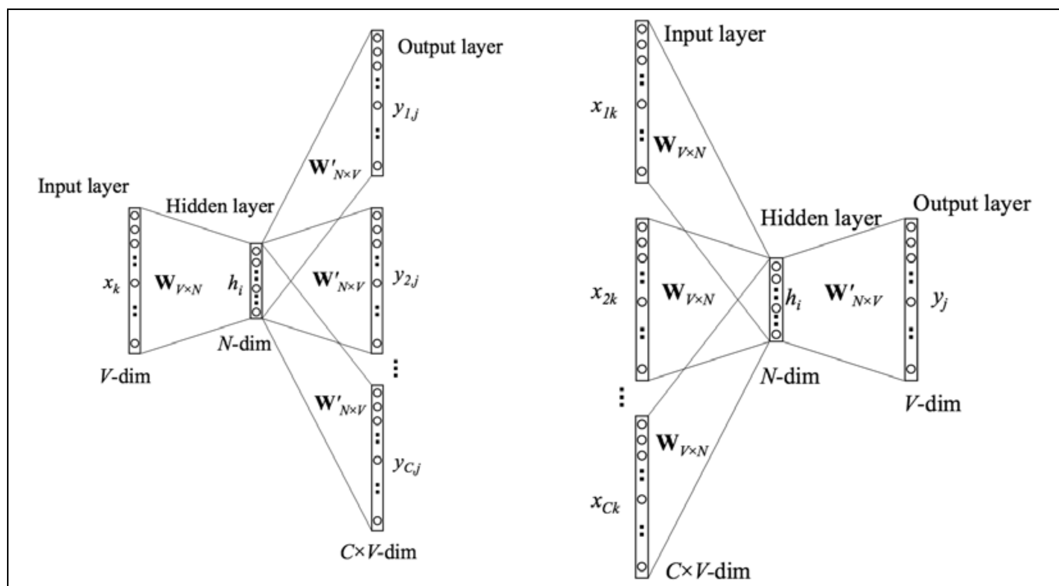


图 4.1 word2vec 算法的 Skip-gram（左）和 CBOW（右）架构；该图摘自 X Rong（2015）所写的“word2vec Parameter Learning Explained”

这两种情况，矩阵 W 将输入向量转换为隐含层，将隐含层 W'转换为输出层的 y，计算得到目标（或上下文）。训练阶段，计算真正的目标单词（或上下文）的错误率，并用随机梯度下降，更新 W 和 W'矩阵。我们将在下一节给出 CBOW 方法更为数学化的表述。请注意，Skip-gram 等式与之类似，我们将参照 Rong（2015）的论文，更为详细地介绍。

5．CBOW 模型的数学表述

模型从输入层开始进入到隐含层 h，隐含层可通过计算 $h=\frac{1}{C}W\cdot(x_1+\cdots+x_C)=\frac{1}{C}(v_{w_1}+\cdots+v_{w_c})=v_C$ 得到，其中，v_{w_i} 向量长度为 N，它表示隐含层单词 w_i。w_c 是 C 上下文向量 v_{w_i} 的均值。选定目标词 w_j，向量 v'_{w_j}（W'的第 j 列）乘以 h 就可以得到输出层 u_j 的值：

$$u_j=v'_{w_j}\cdot h$$

这还不是输出层 y_j 的最终值，因为我们想计算的是给定上下文 C，目标词 w_j 的后验概率，用 **softmax** 公式表示上下文，y_j 的计算方式如下：

$$y_j = p\left(w_j \middle| w_1,.,w_C\right) = \frac{e^{u_j}}{\sum_{i=1}^{|V|} e^{u_i}} = \frac{e^{v_j'^T v_C}}{\sum_{i=1}^{|V|} e^{v_i'^T v_C}}$$

现在，训练目标是，对于词汇表所有单词，最大化这一概率，这等价于 $max\,P\left(w_j \middle| w_1.,w_C\right) \rightarrow E = -\max_j \log p(w_j|w_1,.,w_C) = -v_{w_jM}'^T \bullet h + \log\sum_{i=1}^{|V|} e^{v_{wi}'h}$，其中 $\max_j(v_{w_j}'^T \cdot h) = v_{w_jM}'^T \cdot h$，下标 j^M 表示向量 W'中使得乘积最大化的元素，也就是最可能的目标词。

E 对 $W(w_{ij})$ 和 $W'(w'_{ij})$求偏导，可得到随机梯度下降公式。作为输出的每个目标词 w_j，其计算公式如下：

$$v_{w_j}'^{new} = v_{w_j}'^{old} - \alpha\frac{\partial E}{\partial u_j} h \forall j \in 1,\cdots|V|$$

$v_{w_j}'^{new} = v_{w_j}'^{old} - \alpha\frac{1}{C}\frac{\partial E}{\partial h}\forall j \in 1,\cdots,C$，其中 $\frac{\partial E}{\partial h} = \left(\frac{\partial E}{\partial h_1},\ldots,\frac{\partial E}{\partial h_N}\right)$，α 为梯度下降的学习速率。导数 $\frac{\partial E}{\partial u_j} = y_j - \delta_{j,j^M}$ 表示网络与实际目标词之间的错误率。在系统中反向传播错误率，实现在迭代中学习。注意 $v_{w_j}\forall j1,\cdots,|V|$ 是语义计算常用的词向量（word representations）。

更多细节见 Rong（2015）的论文。

6．Doc2Vec 扩展

Le 和 Mikolov 在论文（2014）中解释过，Doc2Vec 是 word2vec 方法自然而然的扩展，Doc2Vec 将文档看作是额外的词向量。因此，对于 CBOW 架构，隐含层向量 h 为上下文向量和文档向量 d_i 的均值：

$$\begin{aligned} h &= \frac{1}{C}W \cdot (x_1 + \cdots + x_C + d_i) \\ &= \frac{1}{C}(v_{w_1} + \cdots + v_{w_c} + v_{d_i}) = v_C \end{aligned}$$

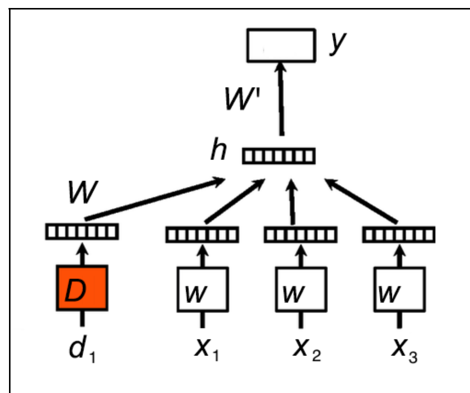


图 4.2　分布记忆模型示例，上下文包含 3 个词（window=3）；该图摘自 Le 和 Mikolov（2014）所写的“*Distributed Representations of Sentences and Documents*”（句子和文档的分布式表示）

图 4.2 所示的架构叫作**分布记忆模型**（Distributed Memory Model，DM），因为文档 d_i 向量记住了上下文词语没有表示出来的文档信息。从文档 d_i 采样

得到（sample）的所有上下文词共用 v_{d_j}，但是对于所有文档，矩阵 W（和 W'）都是相同的。

另一种架构叫作**分布词袋**（Distributed Bag of Words，DBOW），输入层仅考虑一篇文档向量，输出层给出从文档采样得到的一组上下文词语。经验表明 DM 架构比 DBOW 架构效果好，因此 DM 是 gensim 库默认使用的模型。建议读者读一下 Le 和 Mikolov 在 2014 年发表的论文，了解更多细节。

7．影评查询示例

我们使用 Bo Pang 和 Lillian Lee 提供的 IMBD 影评数据，演示前面讨论的 3 种信息检索模型实际用法。数据下载地址为 http://www.cs.cornell.edu/people/pabo/movie-review-data/，请下载 *polarity dataset v2.0* 和 *Pool of 27886 unprocessed html files* 两个文件。（数据集和代码也可以从作者的 GitHub 主页下载，地址为 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_4/）。从网站下载 movie.zip（实际下载下来的文件叫作 polarity_html.zip），解压后生成 movie 文件夹，里面存放了所有的影评网页（大约 2000 个文件）。首先，读取这些文件，准备数据：

```
In [1]: #import files
        import os
        import numpy as np
        #get titles
        from BeautifulSoup import BeautifulSoup
        moviehtmldir = './movie/'
        moviedict = {}
        for filename in [f for f in os.listdir(moviehtmldir) if f[0]!='.']:
            id = filename.split('.')[0]
            f = open(moviehtmldir+'/'+filename)
            parsed_html = BeautifulSoup(f.read())
            try:
                title = parsed_html.body.h1.text

            except:
                title = 'none'
            moviedict[id] = title
```

然后，我们用 **BeautifulSoup** 从每个 HTML 页面解析出网页的标题（title），创建 moviedict 字典。polarity dataset v2.0.tar.gz 包含 review_polarity 文件夹，该文件夹下有 txt_sentoken 文件夹，它的两个子文件夹分别存放着积极和消极影评（pros 和 cons）。用下面代码预处理这些文件：

```
In [99]: import nltk
         from nltk.corpus import stopwords
         from nltk.tokenize import WordPunctTokenizer
         tknzr = WordPunctTokenizer()
         nltk.download('stopwords')
         stoplist = stopwords.words('english')
         from nltk.stem.porter import PorterStemmer
         stemmer = PorterStemmer()
         def ListDocs(dirname):
             docs = []
             titles = []
             for filename in [f for f in os.listdir(dirname) if str(f)[0]!='.']:
                 f = open(dirname+'/'+filename,'r')
                 id = filename.split('.')[0].split('_')[1]
                 titles.append(moviedict[id])
                 docs.append(f.read())
             return docs,titles

         dir = './review_polarity/txt_sentoken/'
         pos_textreviews,pos_titles = ListDocs(dir+'pos/')
         neg_textreviews,neg_titles = ListDocs(dir+'neg/')
         tot_textreviews = pos_textreviews+neg_textreviews
         tot_titles = pos_titles+neg_titles
```

现在，我们已将 2000 条影评存储到 tot_textreviews 列表，网页标题存储到 tot_titles 列表。我们用 sklearn 的 TF-IDF 实现，训练 TF-IDF 模型：

```
In [4]: #test tf-idf
        from sklearn.feature_extraction.text import TfidfVectorizer

        def PreprocessTfidf(texts,stoplist=[],stem=False):
            newtexts = []
            for text in texts:
                if stem:
                    tmp = [w for w in tknzr.tokenize(text) if w not in stoplist]
                else:
                    tmp = [stemmer.stem(w) for w in [w for w in tknzr.tokenize(text) if w not in stoplist]]
                newtexts.append(' '.join(tmp))
            return newtexts
        vectorizer = TfidfVectorizer(min_df=1)
        processed_reviews = PreprocessTfidf(tot_textreviews,stoplist,True)
        mod_tfidf = vectorizer.fit(processed_reviews)
        vec_tfidf = mod_tfidf.transform(processed_reviews)
        tfidf = dict(zip(vectorizer.get_feature_names(),vectorizer.idf_))
```

上述代码 PreprocessTfidf 函数后，预处理每篇文档（删除停用词、分词和提取词干）。同理，用 gensim 库的 LSA 实现，训练 LSA 模型，指定 10 个潜在维度（latent dimension）：

```
In [5]: #test LSA
        import gensim
        from gensim import models
        class GenSimCorpus(object):
                   def __init__(self, texts, stoplist=[],stem=False):
                       self.texts = texts
                       self.stoplist = stoplist
                       self.stem = stem
                       self.dictionary = gensim.corpora.Dictionary(self.iter_docs(texts, stoplist))


                   def __len__(self):
                       return len(self.texts)
                   def __iter__(self):
                       for tokens in self.iter_docs(self.texts, self.stoplist):
                           yield self.dictionary.doc2bow(tokens)
                   def iter_docs(self,texts, stoplist):
                       for text in texts:
                           if self.stem:
                              yield (stemmer.stem(w) for w in [x for x in tknzr.tokenize(text) if x not in stoplist])
                           else:
                              yield (x for x in tknzr.tokenize(text) if x not in stoplist)

        corpus = GenSimCorpus(tot_textreviews,stoplist,True)
        dict_corpus = corpus.dictionary
        ntopics = 10
        lsi =  models.LsiModel(corpus, num_topics=ntopics, id2word=dict_corpus)
```

注意 GenSimCorpus 函数用常用技术预处理文档，将文档转换为 gensim 的 LSA 实现可以读取的格式。我们能够从 lsi 对象获取到 U、V 和 S 矩阵，将查询词转换到潜在空间需要用到的这些矩阵：

```
In [6]: U = lsi.projection.u
        Sigma = np.eye(ntopics)*lsi.projection.s
        #calculate V
        V = gensim.matutils.corpus2dense(lsi[corpus], len(lsi.projection.s)).T / lsi.projection.s
        dict_words = {}
        for i in range(len(dict_corpus)):
            dict_words[dict_corpus[i]] = i
```

经过计算，我们得到了单词的索引字典 dict_words，将查询词转换为它在 dict_corpus

相应的索引词。

最后训练模型是 Doc2Vec。首先，按照 gensim 的 Doc2Vec 实现能够处理的格式，准备好数据：

```
In [7]: from collections import namedtuple

        def PreprocessDoc2Vec(text,stop=[],stem=False):
            words = tknzr.tokenize(text)
            if stem:
                words_clean = [stemmer.stem(w) for w in [i.lower() for i in words if i not in stop]]
            else:
                words_clean = [i.lower() for i in words if i not in stop]
            return words_clean

        Review = namedtuple('Review','words tags')
        dir = './review_polarity/txt_sentoken/'
        do2vecstem = False
        reviews_pos = []
        cnt = 0
        for filename in [f for f in os.listdir(dir+'pos/') if str(f)[0]!='.']:
            f = open(dir+'pos/'+filename,'r')
            reviews_pos.append(Review(PreprocessDoc2Vec(f.read(),stoplist,do2vecstem),['pos_'+str(cnt)]))
            cnt+=1

        reviews_neg = []
        cnt= 0
        for filename in [f for f in os.listdir(dir+'neg/') if str(f)[0]!='.']:
            f = open(dir+'neg/'+filename,'r')
            reviews_neg.append(Review(PreprocessDoc2Vec(f.read(),stoplist,do2vecstem),['neg_'+str(cnt)]))
            cnt+=1

        tot_reviews = reviews_pos + reviews_neg
```

将每条评论置于一个 namedtuple 对象之中，该对象包含 PreprocessDoc2Vec 函数（删除停用词并分词）处理过的单词和表示文件名称的标签。请注意，我们没有提取词干，因为我们发现不提取词干效果更好（读者可将布尔型标记 doc2vecstem 置为 True，试试词干的效果如何）。用以下代码完成 Doc2Vec 训练：

```
In [8]: #define doc2vec
        from gensim.models import Doc2Vec
        import multiprocessing

        cores = multiprocessing.cpu_count()
        vec_size = 500
        model_d2v = Doc2Vec(dm=1, dm_concat=0, size=vec_size, window=10, negative=0, hs=0, min_count=1, workers=cores)

        #build vocab
        model_d2v.build_vocab(tot_reviews)
        #train
        numepochs= 20
        for epoch in range(numepochs):
            try:
                print 'epoch %d' % (epoch)
                model_d2v.train(tot_reviews)
                model_d2v.alpha *= 0.99
                model_d2v.min_alpha = model_d2v.alpha
            except (KeyboardInterrupt, SystemExit):
                break
```

我们设置好用 DM 架构（dm=1），隐含层为 500 维（大小），窗口大小为 10 个单词，将所有至少出现 1 次的单词（min_count=1）纳入模型。其余参数与效率优化方法有关（negative 指的是负采样，hs 指的是 hierarchical softmax，也称为层次 softmax）。训练持续了 20 个 epoch（步数），学习速率为 0.99。

我们现在可以验证每种方法的返回结果。比如定义如下查询词，检索所有跟科幻电影相关的网页文档。这里所说的科幻电影也就是通常能用下面列表中的单词描述的电影：

```
In [9]: #query
        query = ['science','future','action']
```

用 TF-IDF 模型返回 5 个与查询词最相似的网页，代码如下：

```
In [10]: #similar tfidf
         #sparse matrix so the metrics transform into regular vectors before computing cosine
         from sklearn.metrics.pairwise import cosine_similarity
         query_vec = mod_tfidf.transform(PreprocessTfidf([' '.join(query)],stoplist,True))
         sims= cosine_similarity(query_vec,vec_tfidf)[0]
         indxs_sims = sims.argsort()[::-1]
         for d in list(indxs_sims)[:5]:
             print 'sim:',sims[d],' title:',tot_titles[d]
```

模型使用稀疏矩阵存储数据，因此用 cosine_similarity 函数将稀疏向量转换为常规向量，这点要注意。然后，再计算相似度。LSA 模型处理方式类似，将查询词转换为 q_k，然后输出 5 个最相似的网页：

```
In [11]: #LSA query
         def TransformWordsListtoQueryVec(wordslist,dict_words,stem=False):
             q = np.zeros(len(dict_words.keys()))
             for w in wordslist:
                 if stem:
                     q[dict_words[stemmer.stem(w)]]=1.
                 else:
                     q[dict_words[w]] = 1.
             return q

         q = TransformWordsListtoQueryVec(query,dict_words,True)

         qk =   np.dot(np.dot(q,U),Sigma)

         sims = np.zeros(len(tot_textreviews))
         for d in range(len(V)):
             sims[d]=np.dot(qk,V[d])
         indxs_sims = np.argsort(sims)[::-1]
         for d in list(indxs_sims)[:5]:
             print 'sim:',sims[d],' doc:',tot_titles[d]
```

最后，doc2vec 模型用 infer_vector 函数将查询词列表转换为向量。most_similar 函数返回最相似的影评：

```
In [12]: #doc2vec query
         #force inference to get the same result
         model_d2v.random = np.random.RandomState(1)
         query_docvec = model_d2v.infer_vector(PreprocessDoc2Vec(' '.join(query),stoplist,do2vecstem))

         reviews_related = model_d2v.docvecs.most_similar([query_docvec], topn=5)
         for review in reviews_related:
             print 'relevance:',review[1],'  title:',tot_titles[review[0]]
```

请注意，若要用优化方法（负采样或层次 softmax），模型的 random 参数需设置为一个固定值，以返回确定的结果。结果如下：

- TF-IDF：

```
sim: 0.177948650457  title: No Telling (1991)
sim: 0.177821146567  title: Total Recall (1990)
sim: 0.173783798661  title: Time Machine, The (1960)
sim: 0.163031796224  title: Bicentennial Man (1999)
sim: 0.160582512878  title: Andromeda Strain, The (1971)
```

- LSA：

```
sim: 4.0370254245  doc: Star Wars: Episode I - The Phantom Menace (1999)
sim: 3.41798397445  doc: Alien&#179; (1992)
sim: 3.41131742531  doc: Rocky Horror Picture Show, The (1975)
sim: 2.99980957062  doc: Starship Troopers (1997)
sim: 2.86164366049  doc: Wild Things (1998)
```

- Doc2Vec：

```
relevance: 0.129549503326   title: Lost World: Jurassic Park, The (1997)
relevance: 0.124721623957   title: In the Heat of the Night (1967)
relevance: 0.122562259436   title: Charlie's Angels (2000)
relevance: 0.119273915887   title: Batman & Robin (1997)
relevance: 0.118506141007   title: Pok&#233;mon: The Movie 2000 (2000)
```

3 种方法返回的电影中均有和查询词相关的。有趣的是，TF-IDF 算法比更高级的 LSA 和 Doc2Vec 算法效果要好，因为这些算法返回结果中的 *In the Heat of the Night*、*Pokemon*、*Rocky Horror Picture Show* 和 *Wild Things* 与查询词无关，而 TF-IDF 的返回结果仅有一部电影（*No Telling*）与查询词无关。*Charlie's Angels* 和 *Batman & Robin* 这两部是动作片，因此它们与单个查询词 *action* 相关性最大。Doc2Vec 结果最差，主要是因为训练集太小，无法学到好的向量表示，该算法需要较大的训练集（例如，谷歌发布的 word2vec 训练集包含几十亿甚至更多的文档）。康奈尔大学计算机学院网站 http://www.cs.cornell.edu/people/pabo/movie-review-data/提供一个更大的数据集，读者可尝试练习用更多的数据训练 Doc2Vec 模型。

4.4 信息的后处理

从 Web 上收集来网页之后，我们可以用自然语言处理算法从中抽取相关信息，构建 Web 搜索引擎，或用于其他商业目的。我们下面讨论从文档集抽取主题的潜在狄利克雷分析（latent Dirichlet analysis）算法，以及从每个网页抽取情感和观点的技术（观点挖掘技术）。

4.4.1 潜在狄利克雷分配

潜在狄利克雷分配（Latent Dirichlet Allocation，LDA）是一种属于生成模型范畴的自然语言处理算法。该技术是基于以下观察，一些变量可以用潜在、未观察到的变量来解释，这也正是观察到的数据相似或不同的原因。

例如，对于文本文档而言，单词是观察到的变量，而每篇文档可以是多个主题（未观察到的变量）糅合在一起形成的，每个单词指向一特定主题。

例如，下面两篇描述两家公司的文档：

- **文档 1:** Changing how people search for fashion items and, share and buy fashion via visual recognition, TRUELIFE is going to become the best approach to search the ultimate trends …[①]
- **文档 2:** Cinema4you enabling any venue to be a cinema is a new digital filmed media distribution company currently in the testing phase. It applies technologies used in Video on Demand and broadcasting to ... [②]

LDA 可自动发现这些文档所包含的潜在主题。例如，给定以上两篇文档，LDA 也许返回以下与每个主题相关的词语：

- **主题 1**：people Video fashion media…（人们 视频 时尚 媒体……）
- **主题 2**：Cinema technologies recognition broadcasting…（影院 技术 识别 播放……）

因而，可将第 2 个主题标记为技术，第 1 个主题标记为商业。

然后，可将文档表示为多个主题的混合体，文档中的单词以一定的概率分属于不同的主题：

- **文档 1**：主题 1 42%、主题 2 64%
- **文档 2**：主题 1 21%、主题 2 79%

这种文档表示方法可用于多种应用，比如将多个网页分为不同的组或从一组网页中抽取主要的共同主题。下一节，我们解释该算法背后的数学模型。

1．模型

文档表示为潜在主题的随机组合，每个主题用词语的分布来刻画。假定文档集包含 M 篇文档 $d=(d_1,\cdots, d_M)$，每篇文档 i 包含 N_i 个单词。如果 V 是词汇表的长度，文档 i 的每个单词表示为长度为 V 的向量 w_i，其中只有元素 $w_i^v=1$，其余元素为 0：

$$w_i = (0,,, w_i^v=1,\cdots)$$

潜在维度（主题）的数量为 K，对于每篇文章，$z=(z_1,\cdots, z_{N_1})$为每个单词 w_i 对应的主题向

① 译文是“TRUELIFE 正在用视觉识别方法改变人们搜索、分享和购买时尚用品的方式，它即将变为搜索终极潮流的最佳方式……”。因为正文下面涉及单词分属不同主题概率的计算，所以正文仍用英文。下同。——译者注

② 译文是“Cinema4you 是一家新型的数字化影视媒体分销公司，它可将任何会场升级为影院，它目前处于试运营阶段。它采用了视频点播技术，播放……”——译者注

量。z_i向量长度为K，除去第j个元素z_i^j为1，其余元素皆为0，z_i^j表示单词w_i来自的主题。

b为$K×V$矩阵，b_{ij}表示词汇表的每个单词j采自主题i的概率：$\beta_{ij}=p(w^j=1|z^i=1)$。

因此，b的每一行i表示单词在主题i下的分布，而每一列j表示主题在单词j下的分布。我们用以上定义来描述LDA算法：

（1）从选定的分布（通常为泊松），采样得到每篇文档N_i的长度。

（2）对于每篇文档d_i，采样一个主题分布q_i，作为狄利克雷分布$Dir(a)$，其中$i∈1,\cdots M$，α参数是一个长度为K的向量，使得

$$p(\theta_i|\alpha)=\frac{\Gamma\left(\sum_{d=0}^{K}\alpha_d\right)}{\prod_{d=0}^{K}\Gamma(\alpha_d)}\prod_{d=0}^{K}(\theta_i)_d^{\alpha i-1}$$

（3）从多项分布z_n～$Multinominal(\theta_i)$采样文档d_i中单词n的主题。

（4）对于每篇文档d_i、每个单词n和每个主题z_n，从由b的z_n行指定的词频分布w_n～β_{z_n}，采样生成单词w_n。

算法的目标是，对于每篇文档，最大化后验概率：

$$p(\theta_i,z|d_i,\alpha,\beta)=\frac{p(\theta_i,z,d_i|\alpha,\beta)}{p(d_i|\alpha,\beta)}$$

根据条件概率的定义，分子变为：

$$p\left(\theta_i,z,d_i|\alpha,\beta\right)=p\left(d_i|z,\beta\right)p\left(z,|\theta_i\right)p\left(\theta_i|\alpha\right)$$

因此，给定主题向量z和单词概率矩阵b，文档i的概率可以表示为单个单词概率的连乘：

$$p\left(d_i|z,\beta\right)=\prod_{n=1}^{N_i}\beta_{z_n,w_n}$$

由于z_n这个向量，只有第j个元素$z_n^j=1$，其余元素皆为0，那么$p(z|\theta_i)=(\theta_i)_j$。再代入上面的式（2）：[1]

$$p\left(\theta_i,z,d_i|\alpha,\beta\right)=\left(\frac{\Gamma\left(\sum_{d=0}^{K}\alpha_d\right)}{\prod_{d=0}^{K}\Gamma\left(\alpha_d\right)}\prod_{d=0}^{K}\left(\theta_i\right)_d^{\alpha_i-1}\right)\prod_{n=1}^{N_i}\beta_{z_n,w_n}\left(\theta_i\right)_{z_n}$$

① 指$p(Q,2,d_i\mid\alpha,\beta)=p(d_i\mid 2,\beta)\,p(2,|Q_i)\,p(Q_i|\alpha)$。——译者注

（1）[①]式的分母，即文档的边缘分布，对 d_i 积分，对 z 求和就能得到。主题分布 q_i 的最终结果和每个主题分布（b 矩阵的各行）下的单词概率，用近似推断（approximated inference）方法来求解；这些内容超出了本书的范围。

参数 α 叫作集中参数（concentration parameter），它表明分布扩展到可能的值上的程度。集中参数值为 1（或 k，狄利克雷分布的维度，主题模型文献所使用的定义），各组的概率相等。而集中参数取接近 0 的极值时，所得到的分布中几乎所有主题可能都集中到其中一个主题（各单词不再分属于不同的主题，它们分属于少数几个主题）。

举个例子，10 万维的类别分布（categorical distribution），词汇量为 10 万，一个主题也许由几百个单词表示。因此，集中参数标准取值在 0.01 和 0.001 之间，如果词汇量高达百万级甚至更高，那么集中参数的取值还要更小。

根据 L. Li 和 Y. Zhang 的论文 *An empirical study of text classification using Latent Dirichlet Allocation*（用 LDA 为文本分类的实证研究），LDA 可以用作文本模型的降维方法，非常有效。然而，即使该方法在多种应用上表现不错，还是有一些问题要考虑。模型的初始化是随机的，这表明每次运行结果可能不同。此外，集中参数的选取也很重要，但还没有标准化的选取方法。

2．LDA 主题分类示例

我们再次用影评网页数据集 textreviews，在 4.3.1 小节中的“7.影评查询示例”中已预处理过。我们接下来测试 LDA 模型是否能够将影评分成不同的主题。下面代码已经放到我的 GitHub 主页，请从 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_4/下载 postprocessing.ipynb 这个文件。

```
In [6]: #LDA
import gensim.models
from gensim import models

from nltk.tokenize import RegexpTokenizer
tknzr = RegexpTokenizer(r'((?<=[^\w\s])\w(?=[^\w\s])|(\W))+', gaps=True)

from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
class GenSimCorpus(object):
           def __init__(self, texts, stoplist=[],bestwords=[],stem=False):
               self.texts = texts
               self.stoplist = stoplist
               self.stem = stem
               self.bestwords = bestwords
               self.dictionary = gensim.corpora.Dictionary(self.iter_docs(texts, stoplist))

           def __len__(self):
               return len(self.texts)
           def __iter__(self):
               for tokens in self.iter_docs(self.texts, self.stoplist):
                   yield self.dictionary.doc2bow(tokens)
           def iter_docs(self,texts, stoplist):
               for text in texts:
                   if self.stem:
                      yield (stemmer.stem(w) for w in [x for x in tknzr.tokenize(text) if x not in stoplist])
                   else:
                      if len(self.bestwords)>0:
                         yield (x for x in tknzr.tokenize(text) if x in self.bestwords)
                      else:
                         yield (x for x in tknzr.tokenize(text) if x not in stoplist)

num_topics = 10
corpus = GenSimCorpus(tot_textreviews, stoplist,[],False)
dict_lda = corpus.dictionary
```

① 原书未标明的（1）式，指的是最大化后验概率的那个式子。——译者注

我们照旧对每篇文档进行分词（这次使用另一个分词器），并且删除了停用词。为了得到更好的结果，我们过滤掉不会给网页增加任何信息、出现频率最高的词语（比如 movie 和 film）。我们忽略出现次数在 1000 次以上或少于 3 次的单词：

```
In [7]: import copy
        #filter out very common words like mobie and film or very unfrequent terms
        out_ids = [tokenid for tokenid, docfreq in dict_lda.dfs.iteritems() if docfreq > 1000 or docfreq < 3 ]
        dict_lfq = copy.deepcopy(dict_lda)
        dict_lfq.filter_tokens(out_ids)
        dict_lfq.compactify()
        corpus = [dict_lfq.doc2bow(tknzr.tokenize(text)) for text in tot_textreviews]
```

现在，我们可以训练 10 个主题的 LDA 模型（passes 为用整个文档集训练的次数）：

```
In [8]: lda_lfq = models.LdaModel(corpus, num_topics=num_topics, id2word=dict_lfq,passes=10, iterations=50,alpha=0.01,eta=0.01)
        for t in range(num_topics):
            print 'topic ',t,'  words: ',lda_lfq.print_topic(t,topn=10)
            print
```

上述代码返回与每个主题最相关的 10 个单词：

```
topic  0   words:  0.009*best + 0.008*life + 0.008*although + 0.008*great + 0.007*director + 0.006*own + 0.006*see +
0.006*town + 0.006*doesn + 0.005*still

topic  1   words:  0.014*see + 0.010*know + 0.008*bad + 0.008*off + 0.008*think + 0.007*plot + 0.007*could +
0.007*re + 0.007*life + 0.007*m

topic  2   words:  0.011*disney + 0.009*off + 0.009*action + 0.009*plot + 0.008*love + 0.008*life + 0.007*wild +
0.007*could + 0.006*mulan + 0.006*new

topic  3   words:  0.009*scene + 0.008*life + 0.007*new + 0.007*know + 0.007*doesn + 0.007*off + 0.007*could +
0.006*bad + 0.006*director + 0.006*see

topic  4   words:  0.014*truman + 0.009*life + 0.009*best + 0.008*doesn + 0.007*scene + 0.007*own + 0.007*world +
0.007*sandler + 0.007*see + 0.006*new

topic  5   words:  0.009*bad + 0.008*big + 0.008*off + 0.007*plot + 0.007*doesn + 0.007*director + 0.007*scene +
0.007*go + 0.006*see + 0.006*better

topic  6   words:  0.013*plot + 0.012*action + 0.012*alien + 0.011*bad + 0.009*new + 0.008*off + 0.008*planet +
0.008*see + 0.007*could + 0.006*scene

topic  7   words:  0.013*action + 0.009*plot + 0.007*war + 0.007*off + 0.007*see + 0.007*re + 0.007*van +
0.006*director + 0.006*great + 0.006*made

topic  8   words:  0.012*love + 0.009*best + 0.008*see + 0.007*could + 0.007*life + 0.006*new + 0.006*scene +
0.006*off + 0.006*go + 0.006*re

topic  9   words:  0.016*life + 0.010*world + 0.007*scene + 0.007*could + 0.006*mother + 0.006*own + 0.006*love +
0.006*role + 0.006*off + 0.006*father
```

虽然，生成的单词不是都能很好地阐释这 10 个主题，但我们显然可以看到主题 2 的几个单词 disney、mulan（Disney 拍的一部电影）、love 和 life 表明主题 2 跟动画影片相关，而主题 6 的 action、alien、bad 和 planet 单词与科幻电影相关。实际上，我们可以查询所有电影中最可能属于主题 6 的：

```
In [9]: #topics for each doc
        def GenerateDistrArrays(corpus):
                for i,dist in enumerate(corpus[:10]):
                    dist_array = np.zeros(num_topics)
                    for d in dist:
                        dist_array[d[0]] =d[1]
                    if dist_array.argmax() == 6 :
                       print tot_titles[i]
        corpus_lda = lda_lfq[corpus]
        GenerateDistrArrays(corpus_lda)
```

返回结果如下：

```
Rock Star (2001)
Star Wars: Episode I - The Phantom Menace (1999)
Zoolander (2001)
Star Wars: Episode I - The Phantom Menace (1999)
Matrix, The (1999)
Volcano (1997)
Return of the Jedi (1983)
Daylight (1996)
Blues Brothers 2000 (1998)
Alien&#179; (1992)
Fallen (1998)
Planet of the Apes (2001)
```

大多数电影显然是科幻和幻想片，因此 LDA 算法将它们聚到一起是正确的。

请注意主题空间（lda_lfq[corpus]）里的文档表示（document representation），我们还可以用聚类算法（详见第 2 章）处理，我们将其留给读者练手，这里不再赘述。此外，还要注意，由于模型的初始化是随机的，每次运行 LDA 算法，其结果也许不同（如果你得到的结果跟书中不同，也很正常）。

4.4.2 观点挖掘（情感分析）

观点挖掘或情感分析这一领域研究如何抽取当事人的观点，这些观点通常分为积极或消极（或中性）的。这类分析非常实用，在营销领域尤其如此，我们可据此找出公众对产品或服务的观点。观点挖掘的标准方法是将情感（或极性）、积极或消极作为分类问题的目标类别。文档集的特征数为词汇表所有不同单词的数量。分类器通常用 SVM 和朴素贝叶斯。举个例子，我们可以用上面测试 LDA 算法和信息检索模型所用的 2000 篇影评作为数据集，该数据集已标注了类别（积极或消极）。本节的所有代码在 postprocessing.ipynb 文件中，可从 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_4/下载。我们像之前那样导入并预处理数据：

```
In [10]: import nltk
         from nltk.corpus import stopwords
         from nltk.tokenize import WordPunctTokenizer
         tknzr = WordPunctTokenizer()

         from nltk.tokenize import RegexpTokenizer
         tknzr = RegexpTokenizer(r'((?<=[^\w\s])\w(?=[^\w\s])|(\W))+', gaps=True)

         nltk.download('stopwords')
         stoplist = stopwords.words('english')
         from nltk.stem.porter import PorterStemmer
         stemmer = PorterStemmer()


         from collections import namedtuple
```

```
def PreprocessReviews(text,stop=[],stem=False):
    #print profile
    words = tknzr.tokenize(text)
    if stem:
       words_clean = [stemmer.stem(w) for w in [i.lower() for i in words if i not in stop]]
    else:
       words_clean = [i.lower() for i in words if i not in stop]
    return words_clean

Review = namedtuple('Review','words title tags')
dir = './review_polarity/txt_sentoken/'
do2vecstem = True
reviews_pos = []
cnt = 0
for filename in [f for f in os.listdir(dir+'pos/') if str(f)[0]!='.']:
    f = open(dir+'pos/'+filename,'r')
    id = filename.split('.')[0].split('_')[1]
    reviews_pos.append(Review(PreprocessReviews(f.read(),stoplist,do2vecstem),moviedict[id],['pos_'+str(cnt)]))
    cnt+=1

reviews_neg = []
cnt= 0
for filename in [f for f in os.listdir(dir+'neg/') if str(f)[0]!='.']:
    f = open(dir+'neg/'+filename,'r')
    id = filename.split('.')[0].split('_')[1]
    reviews_neg.append(Review(PreprocessReviews(f.read(),stoplist,do2vecstem),moviedict[id],['neg_'+str(cnt)]))
    cnt+=1

tot_reviews = reviews_pos + reviews_neg
```

然后，将数据集切分为训练集（80%）和测试集（20%），将其处理成 nltk 库可以处理的形式（元素为元组的列表或是以文档中的单词为键，以文档对应的标签为值的字典）：

```
In [11]: #split in test training sets
         def word_features(words):
             return dict([(word, True) for word in words])
         negfeatures = [(word_features(r.words), 'neg') for r in reviews_neg]
         posfeatures = [(word_features(r.words), 'pos') for r in reviews_pos]
         portionpos = int(len(posfeatures)*0.8)
         portionneg = int(len(negfeatures)*0.8)
         print portionpos,'-',portionneg
         trainfeatures = negfeatures[:portionneg] + posfeatures[:portionpos]
         print len(trainfeatures)
         testfeatures = negfeatures[portionneg:] + posfeatures[portionpos:]
         #shuffle(testfeatures)
```

现在，我们可以用 nltk 库的 NaiveBayesClassifier 分类器（多项式朴素贝叶斯），训练和测试分类器，并检验错误率：

```
In [12]: from nltk.classify import NaiveBayesClassifier
         #training naive bayes
         classifier = NaiveBayesClassifier.train(trainfeatures)
         ##testing
         err = 0
         print 'test on: ',len(testfeatures)
         for r in testfeatures:
             sent = classifier.classify(r[0])
             if sent != r[1]:
                err +=1.
         print 'error rate: ',err/float(len(testfeatures))
```

上述代码得到 28.25%的错误率，但是用每篇文档的最佳二元组，可以改进结果。二元组指一组连续出现的单词，我们用卡方检验 X^2 寻找不是偶然而是频繁共现的二元组。这些特殊的二元组包含文档的相关信息，用自然语言处理的术语来讲，它们叫作搭配（collocation）。例如，给定由两个词 **w1** 和 **w2** 组成的二元组，语料库中共有 N 个可能的二元组。我们给出零假设 **w1** 和 **w2** 是否出现彼此不相干。我们可以将二元组的出现次数（**w1, w2**）和其余可能的二元组的出现次数用矩阵 O 来表示，见表 4.1：

表 4.1

	w1	非 w1
w2	10	901
非 w2	345	1111111

那么，X^2 的计算方法为 $X^2 = \frac{\sum_{i=0,j=0}^{1,1}(O_{ij}-E_{ij})^2}{E_{ij}}$，$O_{ij}$ 表示由单词（i, j）组成的二元组的出现次数（因此 O_{00}=10），E_{ij} 为二元组（i, j）期望出现的次数（例如，$E_{00} = \left(\frac{(O_{00}+O_{01})}{N} + \frac{(O_{00}+O_{10})}{N}\right)N$）。凭直觉不难看出，观察到的频数 O_{ij} 与期望平均数差别越大，X^2 值越高，因此可以拒绝零假设。能够成为搭配的二元组比起按照期望的平均数出现的二元组，包含更多信息。X^2 值可用 f test（也叫作**均方列联系数**[①]）乘以二元组总数 N 得到，如下所示：

$$X^2 = N\phi, \phi = \frac{O_{00}O_{11} - O_{01}O_{10}}{\sqrt{(O_{10}+O_{11})(O_{01}+O_{00})(O_{01}+O_{11})(O_{00}+O_{10})}}$$

关于搭配和 X^2 检验的更多内容，请参考 C. D. Manning 和 H. Schuetze（1999）合著的 *Foundations of Statistical Natural Language Processing*（《统计自然语言处理基础》）。顺便提一下，作为信息增益的一种度量方法，我们可将 X^2 看作是一种我们在第 3 章定义的特征选择方法。我们借助 nltk 库，用 X^2 方法选择每篇文档的 500 个最佳二元组特征，再次训练朴素贝叶斯分类器，代码如下：

```
In [16]: import itertools
         from nltk.collocations import BigramCollocationFinder
         from nltk.metrics import BigramAssocMeasures
         from random import shuffle

         #train bigram:
         def bigrams_words_features(words, nbigrams=200,measure=BigramAssocMeasures.chi_sq):
             bigram_finder = BigramCollocationFinder.from_words(words)
             bigrams = bigram_finder.nbest(measure, nbigrams)
             return dict([(ngram, True) for ngram in itertools.chain(words, bigrams)])

         negfeatures = [(bigrams_words_features(r.words,500), 'neg') for r in reviews_neg]
         posfeatures = [(bigrams_words_features(r.words,500), 'pos') for r in reviews_pos]
         portionpos = int(len(posfeatures)*0.8)
         portionneg = int(len(negfeatures)*0.8)
         print portionpos,'-',portionneg
         trainfeatures = negfeatures[:portionpos] + posfeatures[:portionneg]
         print len(trainfeatures)
         classifier = NaiveBayesClassifier.train(trainfeatures)
         ##test bigram
         testfeatures = negfeatures[portionneg:] + posfeatures[portionpos:]
         shuffle(testfeatures)
         err = 0
         print 'test on: ',len(testfeatures)
         for r in testfeatures:
             sent = classifier.classify(r[0])
             #print r[1],'-pred: ',sent
             if sent != r[1]:
                 err +=1.
         print 'error rate: ',err/float(len(testfeatures))
```

① 英文为“mean square contingency coefficient”。——译者注

这一次错误率降到 20%，低于常规方法的错误率。X^2 检验还可用来从全部语料中抽取信息量最大的单词。我们可以度量单个单词的词频与它在积极（或消极）文档中词频之间的差距，为单词的重要性打分（例如，如果单词 great 在积极评论中的 X^2 值高于它在消极评论的 X^2 值，这表明该词能够给出评论是积极的这一信息）。我们分别计算每个单词在全部语料、积极语料和消极语料的词频，抽取全部语料中 1 万个最重要的单词，代码如下：

```
In [21]: import nltk.classify.util, nltk.metrics
         tot_poswords = [val for l in [r.words for r in reviews_pos] for val in l]
         tot_negwords = [val for l in [r.words for r in reviews_neg] for val in l]
         from nltk.probability import FreqDist, ConditionalFreqDist
         word_fd = FreqDist()
         label_word_fd = ConditionalFreqDist()

         for word in tot_poswords:
             word_fd[word.lower()] +=1
             label_word_fd['pos'][word.lower()] +=1

         for word in tot_negwords:
             word_fd[word.lower()] +=1
             label_word_fd['neg'][word.lower()] +=1
         pos_words = len(tot_poswords)
         neg_words = len(tot_negwords)

         tot_words = pos_words + neg_words
         #select the best words in terms of information contained in the two classes pos and neg
         word_scores = {}

         for word, freq in word_fd.iteritems():
             pos_score = BigramAssocMeasures.chi_sq(label_word_fd['pos'][word],
                         (freq, pos_words), tot_words)
             neg_score = BigramAssocMeasures.chi_sq(label_word_fd['neg'][word],
                         (freq, neg_words), tot_words)
             word_scores[word] = pos_score + neg_score
         print 'total: ',len(word_scores)
         best = sorted(word_scores.iteritems(), key=lambda (w,s): s, reverse=True)[:10000]
         bestwords = set([w for w, s in best])
```

现在，我们只用每篇文章中最具区分性的单词——bestwords 里面的单词训练朴素贝叶斯分类器：

```
In [22]: #training naive bayes with chi square feature selection of best words
         def best_words_features(words):
             return dict([(word, True) for word in words if word in bestwords])

         negfeatures = [(best_words_features(r.words), 'neg') for r in reviews_neg]
         posfeatures = [(best_words_features(r.words), 'pos') for r in reviews_pos]
         portionpos = int(len(posfeatures)*0.8)
         portionneg = int(len(negfeatures)*0.8)
         print portionpos,'-',portionneg
         trainfeatures = negfeatures[:portionpos] + posfeatures[:portionneg]
         print len(trainfeatures)
         classifier = NaiveBayesClassifier.train(trainfeatures)
         ##test with feature chi square selection
         testfeatures = negfeatures[portionneg:] + posfeatures[portionpos:]
         shuffle(testfeatures)
         err = 0
         print 'test on: ',len(testfeatures)
         for r in testfeatures:
             sent = classifier.classify(r[0])
             #print r[1],'-pred: ',sent
             if sent != r[1]:
                err +=1.
         print 'error rate: ',err/float(len(testfeatures))
```

错误率为 12.75%，鉴于我们数据集相对较小，这个错误率可以说是相当低了。如要保证结果更加可靠，应使用交叉检验方法（见第 3 章），请读者自行练习。我们还可以用 Doc2Vec

向量（4.3 节“7.影评查询示例”中计算过）训练分类器。假定，我们已得到 Doc2Vec 向量，并完成训练，将其存储到 model_d2v.docvecs 对象之中，我们按照先前做法将数据集切分为训练集（80%）和测试集（20%）：

```
In [23]: #split train,test sets
trainingsize = 2*int(len(reviews_pos)*0.8)

train_d2v = np.zeros((trainingsize, vec_size))
train_labels = np.zeros(trainingsize)
test_size = len(tot_reviews)-trainingsize
test_d2v = np.zeros((test_size, vec_size))
test_labels = np.zeros(test_size)

cnt_train = 0
cnt_test = 0
for r in reviews_pos:
    name_pos = r.tags[0]
    if int(name_pos.split('_')[1])>= int(trainingsize/2.):
        test_d2v[cnt_test] = model_d2v.docvecs[name_pos]
        test_labels[cnt_test] = 1
        cnt_test +=1
    else:
        train_d2v[cnt_train] = model_d2v.docvecs[name_pos]
        train_labels[cnt_train] = 1
        cnt_train +=1

for r in reviews_neg:
    name_neg = r.tags[0]
    if int(name_neg.split('_')[1])>= int(trainingsize/2.):
        test_d2v[cnt_test] = model_d2v.docvecs[name_neg]
        test_labels[cnt_test] = 0
        cnt_test +=1
    else:
        train_d2v[cnt_train] = model_d2v.docvecs[name_neg]
        train_labels[cnt_train] = 0
        cnt_train +=1
```

然后，训练 SVM 分类器（**径向基内核 RBF**）或对数几率回归模型：

```
In [27]: #train log regre
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(train_d2v, train_labels)
print 'accuracy:',classifier.score(test_d2v,test_labels)

from sklearn.svm import SVC
clf = SVC()
clf.fit(train_d2v, train_labels)
print 'accuracy:',clf.score(test_d2v,test_labels)
```

逻辑回归和 SVM 正确率非常低，分别为 0.5172 和 0.5225，主要原因在于数据集非常小，无法训练包含多个参数的算法，比如神经网络。

4.5 小结

本章讨论了 Web 数据挖掘最常用和最先进的算法，介绍了如何用多种 Python 库来实现它们。学完本章，你应该已经清楚地理解 Web 数据挖掘领域的难点，具备用 Python 语言处理其中一些较为棘手的问题的能力。下一章，我们将讨论当前商业领域所使用的、最为重要的推荐系统算法。

第 5 章 推荐系统

只要是可选的产品或服务较多，用户无法在合理的时间范围内评价它们的好坏，自然就有使用推荐系统的必要。推荐引擎可以帮助线上的卖家，从大量与终端用户不相关的备选商品中，找出用户有意购买的商品，因此它是电子商务平台的重要部件。推荐系统的典型应用见于Amazon、Netflix、eBay 和 Google Play 商店，这些产品利用收集到的历史数据，向每位用户推荐他们也许想购买的商品。过去 20 年，人们发明了多种推荐技术，我们重点介绍如今为业界采用、最重要的推荐技术，并指出每种方法的优缺点。这些推荐系统分为**基于内容**的过滤（Content-based Filtering，CBF）和**协同过滤**（Collaborative Filtering，CF）。我们还会讨论其他推荐方法（关联规则、对数似然和混合推荐）及如何用多种不同方法评估推荐方法的正确率。我们用 MovieLens 数据集（http://grouplens.org/datasets/movielens/），它包括 943 名用户对 1682 部电影的评分数据（分数从 1 到 5 共 5 等），总数量有 10 万条。每名用户至少给 20 部电影打过分，每部电影从属于多个类型。本章代码依旧可从 GitHub 下载，文件夹地址 https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_5，代码文件为 rec_sys_methods.ipynb。

讨论推荐算法之前，我们先介绍主要的矩阵和常用的度量标准，以便准备数据集、建立推荐系统。

5.1 效用矩阵

推荐系统用到两类数据：用户和商品。每名用户喜欢特定的几种商品。评分 r_{ij}（1 到 5）将用户 i 和商品 j 联系起来，表示用户喜欢商品的程度。把这些数据收集起来，用矩阵来表示，这样的矩阵叫作效用矩阵（utility matrix）R。矩阵的每一行 i，表示用户 i 为哪些商品打过分；矩阵的每一列 j 表示所有为商品 j 打过分的所有用户。下面例子，我们找到 ml-100k 文件夹中的 u.data 文件（和 u.item 文件，它存储电影名称），将其读入进来，转换为 pandas 的 DataFrame

对象（处理后将得到的效用矩阵保存为 csv 格式，即 utilitymatrix.csv 文件），代码如下：

```
In [34]: import numpy as np
         import pandas as pd
         import copy
         import collections
         from scipy import linalg
         import math
         from collections import defaultdict
```

```
In [35]: #data
         df = pd.read_csv('./data/ml-100k/u.data',sep='\t',header=None)
         #movie list
         df_info = pd.read_csv('./data/ml-100k/u.item',sep='|',header=None)
         movielist = [df_info[1].tolist()[indx]+';'+str(indx+1) for indx in xrange(len(df_info[1].tolist()))]
         nmovies = len(movielist)
         nusers = len(df[0].drop_duplicates().tolist())

         min_ratings = 50
         movies_rated  = list(df[1])
         counts = collections.Counter(movies_rated)
         dfout = pd.DataFrame(columns=['user']+movielist)

         toremovelist = []
         for i in range(1,nusers):
             tmpmovielist = [0 for j in range(nmovies)]
             dftmp =df[df[0]==i]
             for k in dftmp.index:
                 if counts[dftmp.ix[k][1]]>= min_ratings:
                    tmpmovielist[dftmp.ix[k][1]-1] = dftmp.ix[k][2]

                 else:
                    toremovelist.append(dftmp.ix[k][1])

             dfout.loc[i] = [i]+tmpmovielist

         toremovelist = list(set(toremovelist))
         dfout.drop(dfout.columns[toremovelist], axis=1, inplace=True)
         dfout.to_csv('data/utilitymatrix.csv',index=None)
```

前两行代码输出如下：

```
In [38]: df = pd.read_csv('data/utilitymatrix.csv')
         df.head(2)
```

Out[38]:

	user	Toy Story (1995);1	GoldenEye (1995);2	Four Rooms (1995);3	Get Shorty (1995);4	Copycat (1995);5	Twelve Monkeys (1995);7	Babe (1995);8	Dead Man Walking (1995);9	Richard III (1995);10	...	Cool Runnings (1993);1035	Hamlet (1996);1039
0	1	5	3	4	3	3	4	1	5	3	...	0	0
1	2	4	0	0	0	0	0	0	0	2	...	0	0

2 rows × 604 columns

除去第 1 列（列命为 user，表示用户的 ID），每一列的列名包括两部分：电影的名称和它在 MovieLens 数据库中的 ID，这两部分用英文分号隔开。矩阵中，元素 0 表示缺失值，因为用户打过分的电影远少于 1600 部，所以很多元素都是 0。请注意，因为少于 50 个评分的电影已从效用矩阵删除，所以总列数为 604（多于 50 个评分的电影只有 603 部）。推荐系统的目标是预测这些缺失的元素，但为了使用某些预测技术，我们需要为缺失元素赋初始值（插值，imputation）。常见的插值方法有两种：每位用户给出的平均分或每件商品的平均分，这两种方法见下面这个函数：

```
In [4]: def imputation(inp,Ri):
            Ri = Ri.astype(float)
            def userav():
                for i in xrange(len(Ri)):
                    Ri[i][Ri[i]==0] = sum(Ri[i])/float(len(Ri[i][Ri[i]>0]))
                return Ri
            def itemav():
                for i in xrange(len(Ri[0])):
                    Ri[:,i][Ri[:,i]==0] = sum(Ri[:,i])/float(len(Ri[:,i][Ri[:,i]>0]))
                return Ri
            switch = {'useraverage':userav(),'itemaverage':itemav()}
            return switch[inp]
```

本章实现的多个算法都会调用该函数，因此为了后续使用方便，我们这里先行讨论了该函数。此外，本章的效用矩阵 R 为 $N\times M$ 维，表示 N 个用户、M 个商品。由于不同算法重复用相似度度量方法，我们接下来给出几个最通用的定义。

5.2 相似度度量方法

向量 x 和 y 可以是用户（效用矩阵的行）或商品（效用矩阵的列），它们之间常用的相似度度量方法有以下两种：

- 余弦相似度：$s(x,y)=\dfrac{\sqrt{\sum_i x_i y_i}}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$ [①]
- 皮尔逊相关系数：$s(x,y)=\dfrac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_i (x_i-\bar{x})^2}\sqrt{\sum_i (y_i-\bar{y})^2}}$，$x$ 和 y[②]分别为两个向量的均值。

这两种度量方法，若均值为 0，结果碰巧相同。有了这些知识，我们可以开始讨论不同的推荐算法，先讲协同过滤。先说一下，需要计算两个向量之间的相似度时，用下面这个 sim()函数：

```
In [8]: from scipy.stats import pearsonr
        from scipy.spatial.distance import cosine
        def sim(x,y,metric='cos'):
            if metric == 'cos':
               return 1.-cosine(x,y)
            else:#correlation
               return pearsonr(x,y)[0]
```

我们用 SciPy 库实现的函数计算这两种相似度（但要注意，scipy 余弦相似度的定义与我们上面讲的刚好相反，因此需要再用 1 减去 cosine 函数的返回值）。

5.3 协同过滤方法

这类推荐方法背后的思想是，用户喜欢相似用户喜欢的商品。简单来讲，我们所做的基本假设是，与用户 B 相似的用户 A，给商品打的分数很可能与 B 相同。实际操作中，我们可用两种方式实现这一概念：比较不同用户的品位，根据最相似用户的品位，预测某用

① 原书此处公式有误，分子去掉根号。——译者注

② 原书“where x and y are the averages of the two vectors”中的“x”和“y”实则为 $\bar{x}$ 和 $\bar{y}$。——译者注

户的打分（基于记忆）；从用户喜好数据中，抽取打分模式，根据这些模式预测打分（基于模型）。这两种方法都需要大量数据，针对特定用户的推荐效果如何，取决于数据中有多少相似用户。推荐系统的常见问题是**冷启动**，这个问题已有比较充分地研究，相关文献通常建议我们结合使用 CF 和 CBF。MovieLens 这个例子，我们假定数据充足，不存在冷启动问题。CF 算法的其他常见问题还有：

1）可扩展性，因为随着用户和产品数量的增加，计算量也会增加（也许有必要采用并行计算技术）；

2）稀疏的效用矩阵，因为通常用户只为少数商品打分（一般尝试用插值解决这个问题）。

5.3.1 基于记忆的协同过滤

这一类协同过滤方法，利用效用矩阵，计算用户或商品之间的相似度。虽然这些方法存在可扩展性和冷启动问题，但是若效用矩阵较大或非常小时，现在很多商用系统选用的却正是该类方法。我们接下来讨论基于用户的协同过滤和基于商品的协同过滤。

1．基于用户的协同过滤

该算法用 k-NN 方法（见第 3 章）找出与给定用户打分记录相似的用户，对他们的打分取加权平均值，补上当前用户的缺失值。

算法描述如下：

对于任意给定用户 i 及其没有标记过的商品 j：

（1）用相似度度量方法 s，找出 K 个为商品 j 打过分的最相似用户。

（2）对于用户 i 没有打分的每种商品 j，取 K 个用户打分的加权平均值作为用户 i 为 j 打的分数：

$$p_{ij} = r_i + \frac{\sum_{k=0}^{K} s(i,k)(r_{kj} - \overline{r}_k)}{\left|\sum_{k=0}^{K} s(i,k)\right|}$$

$\overline{r}_i$ 和 $\overline{r}_k$ 分别为用户 i 和 k 打的平均分，增加这两项是为了削弱主观性的影响（有的用户打分很慷慨，有的则很挑剔），$s(i,k)$表示相似度，前面已讲过。我们甚至还可以利用用户打分的分散程度规范化预测分数，使分数更具可比性：

$$p_{ij}=\overline{r_i}+\frac{\sigma_i\sum_{k=0}^{K}s(i,k)(r_{kj}-\overline{r_k})/\sigma_k}{\left|\sum_{k=0}^{K}s(i,k)\right|}$$

σ_i 和 σ_k 分别为用户 i 和 k 所打分数的标准差。

该算法以近邻数 K 作为参数，K 的取值通常为 20～50，大多数应用使用该范围的取值即可。以往的实验结果表明，用皮尔逊相关系数得到的结果要好于余弦相似度，这很可能是因为计算皮尔逊相关系数时，减去用户打的平均分，相关性公式让用户更具可比性。下面代码给出每个用户的缺失值的预测值。

其中，u_vec 表示用户打的分数，函数 FindKNeighbours.CalcRating 用前面刚讲过的公式，从 u_vec 找到 K 名相似用户的分数，计算预测分数（没有调整预测分数的分布）。如果效用矩阵过于稀疏，找不到近邻，将用户自己打分的平均分作为预测结果返回。预测值若大于 5 或小于 1，分别将其设置为 5 或 1。

```
In [51]: def CF_userbased(u_vec,K,data,indxs=False):
             def FindKNeighbours(r,data,K):
                 neighs = []
                 cnt=0
                 for u in xrange(len(data)):
                     if data[u,r]>0 and cnt<K:
                        neighs.append(data[u])
                        cnt +=1
                     elif cnt==K:
                        break
                 return np.array(neighs)

             def CalcRating(u_vec,r,neighs):
                 rating = 0.
                 den = 0.
                 for j in xrange(len(neighs)):
                     rating += neighs[j][-1]*float(neighs[j][r]-neighs[j][neighs[j]>0][:-1].mean())
                     den += abs(neighs[j][-1])
                 if den>0:
                     rating = np.round(u_vec[u_vec>0].mean()+(rating/den),0)
                 else:
                     rating = np.round(u_vec[u_vec>0].mean(),0)
                 if rating>5:
                     return 5.
                 elif rating<1:
                     return 1.
                 return rating
             #add similarity col
             data = data.astype(float)
             nrows = len(data)
             ncols = len(data[0])
             data_sim = np.zeros((nrows,ncols+1))
             data_sim[:,:-1] = data
             #calc similarities:
             for u in xrange(nrows):
                 if np.array_equal(data_sim[u,:-1],u_vec)==False: #list(data_sim[u,:-1]) != list(u_vec):
                    data_sim[u,ncols] = sim(data_sim[u,:-1],u_vec,'pearson')
                 else:
                    data_sim[u,ncols] = 0.
             #order by similarity:
             data_sim =data_sim[data_sim[:,ncols].argsort()][::-1]
             #find the K users for each item not rated:
             u_rec = np.zeros(len(u_vec))
             for r in xrange(ncols):
                 if u_vec[r]==0:
                    neighs = FindKNeighbours(r,data_sim,K)
                    #calc the predicted rating
                    u_rec[r] = CalcRating(u_vec,r,neighs)
             if indxs:
                     #take out the rated movies
                     seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
                     u_rec[seenindxs] = -1
                     recsvec = np.argsort(u_rec)[::-1][np.argsort(u_rec)>0]

                     return recsvec
             return u_rec
```

2．基于商品的协同过滤

该方法的基本思想与基于用户的协同过滤相同，只不过它计算的是商品而不是用户的相似度。在大多数情况下，用户的数量比商品的数量要多得多，而商品的相似度可以提前计算，即使新用户（如果用户数量 N 非常大）加进来，商品之前的相似度也不会有较大变化，因此可以用该方法实现扩展性更强的推荐系统。

对于每个用户 i 和每件商品 j，算法描述如下：

（1）用一种相似度度量方法 s，找出与用户 i 打过分的商品最相似的 K 件商品。

（2）计算 K 件商品分数的加权平均值，将其作为预测值：

$$p_{ij} = \frac{\sum_{k=0}^{K} s(jk) r_{ik}}{\left|\sum_{k=0}^{K} s(jk)\right|}$$

计算相似度时，可能得到负值，因此为了让 p_{ij} 有意义（也就是正值），我们只将大于 0 的相似度加总（如果我们只想找出最佳推荐商品，而不是计算准确的分数，则无须考虑商品的顺序）。基于商品的协同过滤，K 取 20～50 的值，通常也能满足大多数应用的需要。

我们用一个类实现该算法：

```
In [32]: class CF_itembased(object):
             def __init__(self,data):
                 #calc item similarities matrix
                 nitems = len(data[0])
                 self.data = data
                 self.simmatrix = np.zeros((nitems,nitems))
                 for i in xrange(nitems):
                     for j in xrange(nitems):
                         if j>=i:#triangular matrix
                            self.simmatrix[i,j] = sim(data[:,i],data[:,j])
                         else:
                            self.simmatrix[i,j] = self.simmatrix[j,i]

             def GetKSimItemsperUser(self,r,K,u_vec):
                 items = np.argsort(self.simmatrix[r])[::-1]
                 items = items[items!=r]
                 cnt=0
                 neighitems = []
                 for i in items:
                     if u_vec[i]>0 and cnt<K:
                        neighitems.append(i)
                        cnt+=1
                     elif cnt==K:
                        break
                 return neighitems

             def CalcRating(self,r,u_vec,neighitems):
                 rating = 0.
                 den = 0.
```

```
            for i in neighitems:
                rating +=  self.simmatrix[r,i]*u_vec[i]
                den += abs(self.simmatrix[r,i])
            if den>0:
                rating = np.round(rating/den,0)
            else:
                rating = np.round(self.data[:,r][self.data[:,r]>0].mean(),0)
            return rating

        def CalcRatings(self,u_vec,K,indxs=False):
            #u_rec = copy.copy(u_vec)
            u_rec = np.zeros(len(u_vec))
            for r in xrange(len(u_vec)):
                if u_vec[r]==0:
                    neighitems = self.GetKSimItemsperUser(r,K,u_vec)
                    #calc predicted rating
                    u_rec[r] = self.CalcRating(r,u_vec,neighitems)
            if indxs:
                #take out the rated movies
                seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
                u_rec[seenindxs]=-1
                recsvec = np.argsort(u_rec)[::-1][np.argsort(u_rec)>0]

                return recsvec
            return u_rec
```

CF_itembased 类的构造函数，计算商品相似度矩阵 simmatrix。每次用 CalcRatings 函数计算用户的缺失值时，要用到该矩阵。GetKSimItemsperUser 函数，找出 K 个近邻：与给定用户（u_vec）最相似的用户。CalcRating 函数实现了前面讨论的加权平均值的计算方法。若未找到近邻，则将分数设置为该商品的平均分。

3．最简单的基于商品的协同过滤——slope one 算法

除了用前面讨论的度量方法计算相似度，还有一种非常简单却很有效的推荐算法。我们可以计算得到矩阵 D，它的每一个元素 d_{ij} 表示商品 i 和 j 的平均差值（average difference）：

$$d_{ij}=\frac{\sum_{k=1}^{K}(r_{ki}-r_{kj})n_{ij}^{k}}{\sum_{k=1}^{N}n_{ij}^{k}}$$

如果用户 k 为商品 i 和 j 都打过分，$n_{ij}^{k}=\begin{cases}1 & \text{如果}r_{ki}，r_{kj}>0\\ 0 & \text{（打分数据缺失）}\end{cases}$ 变量值才为 1，$\sum_{k=1}^{N}n_{ij}^{k}$ 表示为商品 i 和 j 都打过分的用户数。该算法跟我们在*基于商品的协同过滤*一节所讲的相同。对于每名用户 i 和每件商品 j：

（1）找出与商品 j 差值最小的 K 件商品，$d_{j}*,\cdots,d_{j}*=d_{j1},\cdots,d_{jK}$（*表示可能的索引值，但是简单起见，我们将其标记为 1 到 K）。

（2）计算加权平均值，将其作为预测值：

$$p_{ij} = \frac{\sum_{k=1}^{K}(d_{jk} + r_{ik})\sum_{l=1}^{N} n_{jk}^{l}}{\sum_{k=1}^{K}\sum_{l=1}^{N} n_{jk}^{l}}$$

虽然该算法比其他 CF 算法简单多了，但正确率不输于它们，并且计算开销更小，还易于实现。下面 SlopeOne 类的实现方式，非常类似于基于商品的 CF 方法的 CF_itembased 类：

```
In [34]: class SlopeOne(object):
    def __init__(self,Umatrix):
        #calc item similarities matrix
        nitems = len(Umatrix[0])
        self.difmatrix = np.zeros((nitems,nitems))
        self.nratings = np.zeros((nitems,nitems))
        def diffav_n(x,y):
            xy = np.vstack((x, y)).T
            xy = xy[(xy[:,0]>0) & (xy[:,1]>0)]
            nxy = len(xy)
            if nxy == 0:
                #print 'no common'
                return [1000.,0]
            return [float(sum(xy[:,0])-sum(xy[:,1]))/nxy,nxy]

        for i in xrange(nitems):
            for j in xrange(nitems):
                if j>=i:#triangular matrix
                    self.difmatrix[i,j],self.nratings[i,j] = diffav_n(Umatrix[:,i],Umatrix[:,j])
                else:
                    self.difmatrix[i,j] = -self.difmatrix[j,i]
                    self.nratings[i,j] = self.nratings[j,i]

    def GetKSimItemsperUser(self,r,K,u_vec):
        items = np.argsort(self.difmatrix[r])
        items = items[items!=r]
        cnt=0
        neighitems = []
        for i in items:
            if u_vec[i]>0 and cnt<K:
                neighitems.append(i)
                cnt+=1
            elif cnt==K:
                break
        return neighitems

    def CalcRating(self,r,u_vec,neighitems):
        rating = 0.
        den = 0.
        for i in neighitems:
            if abs(self.difmatrix[r,i])!=1000:
                rating +=  (self.difmatrix[r,i]+u_vec[i])*self.nratings[r,i]
                den += self.nratings[r,i]
        if den==0:
            #print 'no similar diff'
            return 0.
        rating = np.round(rating/den,0)
        if rating >5:
            return 5.
        elif rating <1.:
            return 1.
        return rating

    def CalcRatings(self,u_vec,K):
        #u_rec = copy.copy(u_vec)
        u_rec = np.zeros(len(u_vec))
        for r in xrange(len(u_vec)):
            if u_vec[r]==0:
                neighitems = self.GetKSimItemsperUser(r,K,u_vec)
                #calc predicted rating
                u_rec[r] = self.CalcRating(r,u_vec,neighitems)
        return u_rec
```

唯一的区别在于矩阵：difmatrix 前面已解释过，该算法中它是用来计算商品 i 和 j 的差值 $d(i,j)$，函数 GetKSimItemsperUser 寻找 difmatrix 的最小值，以确定 K 个近邻。两件商品

有可能（虽然看似不可能）没有人打过分，而 difmatrix 可以包含没有定义的值，默认将其设置为 1000。预测值可能大于 5 或小于 1，遇到该情况，必须分别将其设置为 5 或 1。

5.3.2 基于模型的协同过滤

该类方法利用效用矩阵生成模型，抽取用户的打分模式。包含各种模式的模型返回预测值，填充或逼近原始矩阵（矩阵分解）。人们研究过各种推荐模型，发表过很多论文。我们集中讨论**矩阵分解**算法——**奇异值分解**（Singular Value Decomposition，SVD，以及最大期望算法）、**交替最小二乘**（Alternating Least Square，ALS）、**随机梯度下降**（Stochastic Gradient Descent，SGD）和**非负矩阵分解**（Non-negative Matrix Factorization，NMF）算法。

1．交替最小二乘（ALS）

这是分解矩阵 R 的最简单方法。每名用户和每件商品可以表示到 K 维的特征空间：

$$R \simeq PQ^T = \hat{R}$$

其中，$N \times K$ 维的矩阵 P 为特征空间新的用户矩阵，$M \times K$ 维的矩阵 Q 为商品在特征空间的映射。因此，这个问题可以简化为最小化正则化后的代价函数 J：

$$J = \min_{p,q} \sum_{ij} e_{ij}^2 = \min_{p,q} \sum_{ij} M_{ij} \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj} \right)^2 + \frac{\lambda}{2} \left(\left| p_i \right|^2 + \left| q_j^T \right|^2 \right)$$

其中，λ 为正则化参数，通过调整学习到的参数，保证向量 p_i 和 q_j^T 的量级不至于过大，以避免过拟合问题。矩阵元素 Mc_{ij} 的值取决于用户 i 是否为商品 j 打过分，如果 $r_{ij}>0$，Mc_{ij} 为 1，否则为 0。对于每个用户向量 p_i 和商品向量 q_j，令 J 的导数为 0，得到以下两个等式：

$$p_i = \left(Q^T Mc_i Q + \frac{\lambda}{2} I \right)^{-1} Q^T Mc_i R_i$$

$$q_j = \left(P^T Mc_j P + \frac{\lambda}{2} I \right)^{-1} P^T Mc_i R_j$$

其中，R_i 和 Mc_i 为 R、Mc 向量的 i 行，R_j 和 Mc_j 为 R、Mc 向量的 j 列。交替固定矩阵 P、Q，上面两个等式，可直接用最小二乘法（ALS）[①]求解。下面这个函数展示了如何用 Python 实现 ALS 算法：

① least square algorithm，也称作最小平方法。——译者注

```
In [12]: def ALS(Umatrix, K, iterations=50, l=0.001, tol=0.001):

             nrows = len(Umatrix)
             ncols = len(Umatrix[0])
             P = np.random.rand(nrows,K)
             Q = np.random.rand(ncols,K)
             Qt = Q.T
             err = 0.
             Umatrix = Umatrix.astype(float)
             mask = Umatrix>0.
             mask[mask==True]=1
             mask[mask==False]=0
             mask = mask.astype(np.float64, copy=False)
             for it in xrange(iterations):
                 for u, mask_u in enumerate(mask):
                     P[u] = np.linalg.solve(np.dot(Qt, np.dot(np.diag(mask_u), Qt.T)) + l*np.eye(K),
                                            np.dot(Qt, np.dot(np.diag(mask_u), Umatrix[u].T))).T
                 for i, mask_i in enumerate(mask.T):
                     Qt[:,i] = np.linalg.solve(np.dot(P.T, np.dot(np.diag(mask_i), P)) + l*np.eye(K),
                                            np.dot(P.T, np.dot(np.diag(mask_i), Umatrix[:,i])))
                 err=np.sum((mask*(Umatrix - np.dot(P, Qt)))**2)
                 if err < tol:
                     break
             return np.round(np.dot(P,Qt),0)
```

矩阵 *Mc* 叫作 mask，变量 l 表示正则化参数 lambda，默认值为 0.001。我们用 NumPy 库的 linalg.solve 函数求解最小二乘问题。该方法通常没有**随机梯度下降**（SGD）和**奇异值分解**（SVD）（见下面几节）精确，但是它易于实现，易于用并行方法计算（因此速度较快）。

2. 随机梯度下降（SGD）

该方法同样从属于矩阵分解类别，因为它依赖于对效用矩阵 R 的近似：

$$R \simeq PQ^T = \hat{R}$$

上式，矩阵 P（$N\times K$）和矩阵 Q（$M\times K$）分别为用户、商品在 K 维潜在特征空间的表征（representation）。每个近似的分数 $\hat{r}_{ij}$ 可以表示为：

$$\hat{r}_{ij} = \sum_{k=1}^{K} p_{ik} q_{kj}$$

用 ALS 算法求解正则均方误（regularized squared errors）e_{ij}^2 的最小化问题，找到矩阵 $\hat{R}$（代价函数 J 同第 3 章）：

$$\min_{p,q} \sum_{ij} e_{ij}^2 = \min_{p,q} \sum_{ij} \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj} \right)^2 + \frac{\lambda}{2} \left(|p_i|^2 + |q_j^T|^2 \right)$$

最小化问题，用梯度下降方法求解（见*第 3 章有监督机器学习*）：

$$p_{ik} = p_{ik} + \alpha \frac{\partial e_{ij}^2}{\partial p_{ik}} = p_{ik} + \alpha(2e_{ij} q_{kj} - \lambda p_{ik})$$

$$p_{kj} = p_{kj} + \alpha \frac{\partial e_{ij}^2}{\partial p_{kj}} = q_{kj} + \alpha(2e_{ij} q_{ik} - \lambda q_{kj})$$

其中，α 为学习速率（见第 3 章），$e_{ij}=\left(r_{ij}-\sum_{k=1}^{K}p_{ik}q_{kj}\right)$。交替计算前面两个等式（固定 q_{kj}，求 p_{ik}，再反对来进行）直到收敛得到 R。SGD 比 SVD（下一节讲）更易于并行处理（因此更快）。SGD 算法的 Python 实现如下所示：

```
In [11]: def SGD(Umatrix, K, iterations=100, alpha=0.00001, l=0.001, tol=0.001):

             nrows = len(Umatrix)
             ncols = len(Umatrix[0])
             P = np.random.rand(nrows,K)
             Q = np.random.rand(ncols,K)
             Qt = Q.T
             cost=-1
             for it in xrange(iterations):
                 for i in xrange(nrows):
                     for j in xrange(ncols):
                         if Umatrix[i][j] > 0:
                            eij = Umatrix[i][j] -np.dot(P[i,:],Qt[:,j])
                            for k in xrange(K):
                                P[i][k] += alpha*(2*eij*Qt[k][j]-l*P[i][k])
                                Qt[k][j] += alpha*(2*eij*P[i][k]-l*Qt[k][j])
                 cost = 0
                 for i in xrange(nrows):
                     for j in xrange(ncols):
                         if Umatrix[i][j]>0:
                            cost += pow(Umatrix[i][j]-np.dot(P[i,:],Qt[:,j]),2)
                            for k in xrange(K):
                                cost += float(l/2.0)*(pow(P[i][k],2)+pow(Qt[k][j],2))
                 if cost < tol:
                       break
             return np.round(np.dot(P,Qt),0)
```

SGD 函数有几个默认参数，学习速率 α=0.0001，正则化参数 λ=l=0.001，最大迭代次数为 1000，收敛判据（convergence tolerance）tol=0.001。同样，要注意的是，未打分的商品（分数为 0），不参与计算。因此，使用该方法，无须事先填充缺失值（插值）。

3. 非负矩阵分解（NMF）

这一组方法同样将 P（$N\times K$）和矩阵 Q（$M\times K$）的积作为矩阵 R 的分解（K 为特征空间的维度），但要求矩阵元素为非负的。相应的最小化问题为：

$$J=\min_{p,q}\sum_{ij}e_{ij}^2=\min_{p\geqslant 0,q\geqslant 0}\frac{1}{2}\sum_{ij}\left(r_{ij}-\sum_{k=1}^{K}p_{ik}q_{kj}\right)^2+(1-a)\frac{\lambda}{2}\left(\left|p_i\right|^2+\left|q_j^T\right|^2\right)+\alpha\lambda\left(\left|p_i\right|+\left|q_j^T\right|\right)$$

参数 α 决定使用哪种正则化方法（0 为平方，1 为套索正则化或两者的混合），λ 为正则化参数。该最小化问题有几种解法，比如投影梯度法（projected gradient）、坐标下降法（coordinate descent）、非负约束最小二乘法（non-negativity constrained least squares）。这些方法的详细讨论超出本书讲解范围，但是我们自己编写函数时，用到了 sklearn NMF[①]实现的坐标下降法：

① 原书误写为“NFM”。——译者注

```
In [13]: from sklearn.decomposition import NMF
         def NMF_alg(Umatrix,K,inp='none',l=0.001):
             R_tmp = copy.copy(Umatrix)
             R_tmp = R_tmp.astype(float)
             #imputation
             if inp != 'none':
                 R_tmp = imputation(inp,Umatrix)
             nmf = NMF(n_components=K,alpha=l)
             P = nmf.fit_transform(R_tmp)
             R_tmp = np.dot(P,nmf.components_)
             return R_tmp
```

分解矩阵之前，可对矩阵插值。函数 fit_transform 返回 P 矩阵，而 Q^T 矩阵存储于 nmf.components_对象。α 的默认值为 0（平方正则化），我们将其设置为 1：alpha=1。λ 的默认值为 0.01：l=0.01。既然效用矩阵元素为正值（分数），这类方法必然是预测分数的好方法。

4．奇异值分解（SVD）

作为特征降维方法，第 2 章已讨论过该算法，我们将矩阵分解为 U、Σ、V（更多技术细节，见第 2 章），用它们来近似表示矩阵。在协同过滤算法中，我们用 SVD 分解矩阵，但首先要用插值方法，估计每位用户的缺失值；最常用的插值方法是，取效用矩阵的每行（或列）或两者的均值（而不是留下 0 值不管）替代缺失值。除了直接用 SVD 算法分解效用矩阵，还可以用最大期望算法（第 2 章），先从矩阵 $\hat{R}=R$ 着手：

（1）最大化步：计算 $\hat{R}=SVD(\hat{R})$

（2）期望步：$\hat{r}_{ij}=\begin{cases} r_{ij} & \text{如果用户打过分数} \\ \hat{r}_{ij} & \text{（打分数据缺失）} \end{cases}$

重复以上两步，直到均方误之和 $\sum_{ij}(r_{ij}-\hat{r}_{ij})^2$ 小于给定的容忍值（tolerance）。该算法及简单的 SVD 分解，代码如下：

```
In [14]: from sklearn.decomposition import TruncatedSVD
         def SVD(Umatrix,K,inp='none'):
             R_tmp = copy.copy(Umatrix)
             R_tmp = R_tmp.astype(float)
             #imputation
             if inp != 'none':
                 R_tmp = imputation(inp,Umatrix)

             means = np.array([ R_tmp[i][R_tmp[i]>0].mean() for i in xrange(len(R_tmp))]).reshape(-1,1)
             R_tmp = R_tmp-means
             svd = TruncatedSVD(n_components=K, random_state=4)
             R_k = svd.fit_transform(R_tmp)
             R_tmp = svd.inverse_transform(R_k)
             R_tmp = means+R_tmp

             return np.round(R_tmp,0)
```

```
In [15]: def SVD_EM(Umatrix,K,inp='none',iterations=50,tol=0.001):
             R_tmp = copy.copy(Umatrix)
             R_tmp = R_tmp.astype(float)
             nrows = len(Umatrix)
             ncols = len(Umatrix[0])
             #imputation
             if inp != 'none':
                 R_tmp = imputation(inp,Umatrix)
             #define svd
             svd = TruncatedSVD(n_components=K, random_state=4)
             err = -1
             for it in xrange(iterations):
                 #m-step
                 R_k = svd.fit_transform(R_tmp)
                 R_tmp = svd.inverse_transform(R_k)
                 #e-step and error evaluation
                 err = 0
                 for i in xrange(nrows):
                     for j in xrange(ncols):
                         if Umatrix[i][j]>0:
                             err += pow(Umatrix[i][j]-R_tmp[i][j],2)
                             R_tmp[i][j] = Umatrix[i][j]

                 if err < tol:
                     print it,'toll reached!'
                     break
             return np.round(R_tmp,0)
```

我们用的是 sklearn 库实现的 SVD 算法，并实现了两种均值插值方法（用户打分的均值和商品得分的均值），虽然我们默认用参数值 *none*，即用 0 值作为缺失值的初始值。最大期望 SVD 算法，其他默认参数有收敛判据（0.0001）和最大迭代次数（10000）。SVD 算法（尤其是用最大期望算法）比 ALS 要慢，但正确率通常更高。还要注意的是，用 SVD 方法分解效用矩阵时，减去了用户打分的均值，因为这样做效果通常更好（SVD 矩阵计算完成后，再加上用户打分的均值）。

本节最后，我们再声明一点，SVD 分解还可用于基于记忆的 CF 方法，在降维后的空间（矩阵 U 或 V^T ）比较用户或商品，然后再根据原来的效用矩阵给出分数（SVD 结合 k-NN 方法）。

5.4 CBF 方法

该类方法从描述商品的数据中抽取用户的特征。MovieLens 这个例子，每部电影 j 都有一组 G 个二进制字段，来表明电影具有以下哪种类型的要素：未知、动作、冒险、动画、儿童、喜剧、犯罪、纪录、剧情、幻想、黑色电影、恐怖、音乐、神秘、浪漫、科幻、惊悚、战争或美国西部片。

我们用这些特征（类型）来刻画电影，将每部电影表示为 G 维（电影类型的数量）的二进制向量 m_j，电影 j 包含哪种类型，向量中代表该类型的元素值为 1，否则为 0。给定存储着效用矩阵 dfout 的 DataFrame 对象（见*效用矩阵*一节），下面代码构造二进制向量 m_j，并将其置于 DataFrame 对象 dfout_movies 之中：

```
In [ ]: #matrix movies's content
        movieslist = [int(m.split(';')[-1]) for m in dfout.columns[1:]]
        moviescats = ['unknown','Action','Adventure','Animation','Children\'s','Comedy','Crime','Documentary',
                      'Drama','Fantasy','Film-Noir','Horror','Musical','Mystery',
                      'Romance','Sci-Fi','Thriller','War','Western']
        dfout_movies =  pd.DataFrame(columns=['movie_id']+moviescats)
        startcatsindx = 5
        cnt= 0
        for m in movieslist:
            dfout_movies.loc[cnt] = [m]+df_info.iloc[m-1][startcatsindx:].tolist()
            cnt +=1
        print dfout_movies.head()

        dfout_movies.to_csv('data/movies_content.csv',index=None)
```

将电影内容矩阵存储到 movies_content.csv 文件，后面 CBF 方法会用到该文件。

基于内容进行推荐的系统，其目标是生成用户画像，用相同字段表明用户喜欢每种类型的程度。该方法的问题是，有时无法描述商品的内容，因此在电子商务环境该方法并不总是可行的。该方法的优点在于，针对特定用户的推荐，独立于其他用户的打分情况，因此不会因为特定商品打分人数不足而受制于冷启动问题。我们将讨论两种推荐方法，从中找出最佳的。方法一，计算每位用户每种类型电影的平均分数，生成用户画像，用余弦相似度度量方法，找出跟用户喜欢的电影最相似的电影。方法二，用正则化线性回归模型，根据用户打分数据和电影特征，生成用户画像特征。用其他用户的画像，预测每位用户尚未看过的电影分数。

5.4.1 商品特征平均得分方法

该方法确实非常简单，我们用 MovieLens 例子描述电影的特征来解释该方法，这些特征前面讲过。该方法的目标是为每位用户 i 生成电影类型喜好向量 $v_i = (v_{i0},\cdots,v_{iG-1})$（长度为 G）。具体要计算平均分 $\overline{r}_i$ 和每种类型 g； v_{ig} 的计算方法是，用户 i（Mi）所看过的包含类型 g 的各部电影的得分，减去平均分之后再加总，然后除以包含类型 g 的电影数量：

$$v_{ig} = \frac{\sum_{k=0}^{M_i}(r_{ik} - \overline{r}_i)I_{kg}}{\sum_{k=0}^{M_i} I_{kg}}$$

其中，如果电影 k 包含类型 g， I_{kg} 为 1，否则为 0。

计算向量 V_i 和二进制向量 m_j 之间的余弦相似度，将相似度最高的电影推荐给用户 i。我们用一个 Python 类实现该方法：

```
In [16]: class CBF_averageprofile(object):
             def __init__(self,Movies,Movieslist):
                 #calc user profiles:
                 self.nfeatures = len(Movies[0])
                 self.Movieslist = Movieslist
                 self.Movies = Movies

             def GetRecMovies(self,u_vec,indxs=False):
                 #generate user profile
                 nmovies = len(u_vec)
                 nfeatures = self.nfeatures
                 mean_u = u_vec[u_vec>0].mean()
                 diff_u = u_vec-mean_u
                 features_u = np.zeros(nfeatures).astype(float)
                 cnts = np.zeros(nfeatures)
                 for m in xrange(nmovies):
                     if u_vec[m]>0:#u has rated m
                         features_u += self.Movies[m]*(diff_u[m])
                         cnts += self.Movies[m]
                 #average:
                 for m in xrange(nfeatures):
                     if cnts[m]>0:
                         features_u[m] = features_u[m]/float(cnts[m])

                 #calc sim:
                 sims = np.zeros(nmovies)
                 for m in xrange(nmovies):
                     if u_vec[m]==0:#sim only for movies not yet rated by the user
                         sims[m] = sim(features_u,self.Movies[m])
                 #order movies
                 order_movies_indxs = np.argsort(sims)[::-1]
                 if indxs:
                     return order_movies_indxs
                 return self.Movieslist[order_movies_indxs]
```

构造器将电影名称存储于 MoviesList 列表，电影特征存储于 Movies 向量，GetRecMovies 函数生成用户喜欢的电影类型向量 v_i（用这一节的公式），也就是代码中的 features_u，然后返回与该向量最相似的电影。

5.4.2　正则化线性回归方法

该方法学习线性模型的参数 θ_i $i \in 0,\cdots,N-1$，$\theta_i \in R^{G+1}$: $\theta_{i0}=1$，N 为用户数，G 为每个商品的特征数（电影类型数）。我们为用户参数 θ_i（$\theta_{i0}=1$）增加截距项（intercept value），同样为电影向量 m_j 增加截距项，$m_{j0}=1$，因此 $m_j \in R^{G+1}$。为了学习参数向量 q_i，需要求解以下带正则项的最小化问题：

$$\min_{\theta_0 \cdots \theta_{N-1}} \frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(\theta_i^T m_j - r_{ij})^2 I_{ij} + \frac{\lambda}{2}\sum_{i=0}^{N-1}\sum_{k=0}^{G-1}\theta_{ik}^2$$

其中，I_{ij} 为 1，表示用户 i 为这部电影打过分，否则 j 为 0，λ 为正则化参数（见第 3 章）。

这个最小化问题可以用梯度下降法求解（见第 3 章）。对于每位用户 i：

- $\theta_{i0} = \theta_{i0} - \alpha\sum_{j=0}^{M-1}(\theta_i^T m_j - r_{ij}) m_{j0} I_{ij} (k=0)$

- $\theta_{ik} = \theta_{ik} - \alpha\left(\sum_{j=0}^{M-1}(\theta_i^T m_j - r_{ij}) m_{j0} I_{ij}\right) m_{j0} I_{ij} + \lambda\theta_{ik} (k>0)$

我们分别为电影和用户向量增加了一截距项，而截距参数（*k=0*）和其他参数的学习，需要区别开来（因为截距不存在过拟合的可能性，不用对其进行正则化处理）。学习到参数 θ_i 之后，对于任何缺失值 r_{ij}，用公式 $r_{ij} = \theta_i^T m_j$ 求解。

该方法的代码实现如下：

```
In [35]: class CBF_regression(object):
             def __init__(self,Movies,Umatrix,alpha=0.01,l=0.0001,its=50,tol=0.001):
                 #calc parameters:
                 self.nfeatures = len(Movies[0])+1#intercept
                 nusers = len(Umatrix)
                 nmovies = len(Umatrix[0])
                 #add intercept col
                 movies_feats = np.ones((nmovies,self.nfeatures))
                 movies_feats[:,1:] = Movies
                 self.movies_feats = movies_feats.astype(float)

                 #set Umatrix as float
                 self.Umatrix = Umatrix.astype(float)
                 #initialize the matrix:
                 Pmatrix = np.random.rand(nusers,self.nfeatures)
                 Pmatrix[:,0]=1.
                 err = 0.
                 cost = -1
                 for it in xrange(its):
                     print 'it:',it,' -- ',cost
```

```
            for u in xrange(nusers):
                for f in xrange(self.nfeatures):
                    if f==0:#no regularization
                        for m in xrange(nmovies):
                            if self.Umatrix[u,m]>0:
                                diff = np.dot(Pmatrix[u],self.movies_feats[m])-self.Umatrix[u,m]
                                Pmatrix[u,f] += -alpha*(diff*self.movies_feats[m][f])
                    else:
                        for m in xrange(nmovies):
                            if self.Umatrix[u,m]>0:
                                diff = np.dot(Pmatrix[u],self.movies_feats[m])-self.Umatrix[u,m]
                                Pmatrix[u,f] += -alpha*(diff*self.movies_feats[m][f] +l*Pmatrix[u][f])

            cost = 0
            for u in xrange(nusers):
                for m in xrange(nmovies):
                    if self.Umatrix[u][m]>0:
                        cost += 0.5*pow(Umatrix[u][m]-np.dot(Pmatrix[u],self.movies_feats[m]),2)
                for f in xrange(1,self.nfeatures):
                    cost += float(l/2.0)*(pow(Pmatrix[u][f],2))
            if cost < tol:
                print 'err',cost
                break
        self.Pmatrix = Pmatrix

    def CalcRatings(self,u_vec):
        #find u_vec
        s = 0.
        u_feats = np.zeros(len(self.Pmatrix[0]))
        #in case the user is not present in the utility matrix find the most similar
        for u in xrange(len(self.Umatrix)):
            #print self.Umatrix[u]
            tmps = sim(self.Umatrix[u],u_vec)
            if tmps > s:
                s = tmps
                u_feats = self.Pmatrix[u]
            if s == 1.:
                break
        new_vec = np.zeros(len(u_vec))
        for r in xrange(len(u_vec)):
            if u_vec[r]==0:
                new_vec[r] = np.dot(u_feats,self.movies_feats[r])
        return new_vec
```

构造器 CBF_regression 用梯度下降方法寻找参数 θ_i（代码中的 Pmatrix），函数 CalcRatings 从效用矩阵 R（用户不在效用矩阵中）找出最相似的分数向量，然后利用相应的参数向量预测缺失值。

5.5 用关联规则学习，构建推荐系统

虽然很多商业推荐系统往往不使用关联规则挖掘，但由于它可以利用历史数据，因此我们应该了解该方法。其实，我们可以用它解决多种实际问题。该方法的主要思想是，统计交易数据库 T 中商品的出现情况，找到商品之间的关系（例如，用户 i 看过的电影或购买的商品，可以看作是一条交易数据）。更正式地讲，一条规则可以是{*item1, item2*}=>{*item3*}这样的形式，即一组商品{*item1, item2*}暗示着另一组商品{*item3*}的存在。每条 X=>Y 规则用以下两个定义来刻画：

- **支持度**（support）：给定一组商品 X，其支持度 $supp(X)$为包含 X 的交易数据占总交易数据的比例。

- **置信度**（confidence）：同时包含 X 和 Y 的交易数据占只包含 X 的比例：$conf(X=>Y)=supp(X \cup Y)/supp(X)$。置信度 $conf(X=>Y)$的值可能与 $conf(Y=>X)$差别较大。

支持度表示一条规则在交易数据库中的频率，而置信度表示 X 出现的情况下，Y 出现的概率。换句话说，我们用支持度筛选我们希望从数据库挖掘的规则（支持度定的越高，满足条件的规则就越少），而置信度可以看作是 X 和 Y 的相似度度量方法。回到电影推荐系统这个例子，交易数据库可以从效用矩阵 R 生成：从每位用户喜欢的电影中，寻找由 X 和 Y 组成的规则，其中集合 X 和 Y 各只包含一个商品（电影）。我们用矩阵 *ass_matrix* 来组织这些规则，矩阵的每个元素 *ass_matrixij* 表示规则 $i=>j$ 的置信度。用 ass_matrix 乘以用户的打分向量 u_vec，就能得到向该用户推荐什么商品：$recitems = uvec \cdot assmatrix$，对 *recitems* 排序，优先推荐 *recitems* 值大的电影。因而，该方法得到的不是对电影打分的预测，而是电影的推荐列表；关联规则方法速度快，即使效用矩阵为稀疏矩阵，也能取得较好的效果。对于如何以最快的速度找到所有可能的商品组合，以得到 X 和 Y，文献中提到了两种方法：*apriori* 和 *fp-growth* 算法（我们不再讨论，因为我们要找的规则中 X、Y 都只包含一个商品）。

下面我们用一个类来实现关联规则推荐：

```
In [36]: class AssociationRules(object):
             def __init__(self,Umatrix,Movieslist,min_support=0.1,min_confidence=0.1,likethreshold=3):
                 self.min_support = min_support
                 self.min_confidence = min_confidence
                 self.Movieslist = Movieslist
                 #transform utility matrix to sets of liked items
                 nitems = len(Umatrix[0])
                 transactions = []
                 for u in Umatrix:
                     s = [i for i in xrange(len(u)) if u[i]>likethreshold]
                     if len(s)>0:
                        transactions.append(s)
                 #find sets of 2 items
                 flat = [item for sublist in transactions for item in sublist]
                 inititems = map(frozenset,[ [item] for item in frozenset(flat)])
                 set_trans = map(set, transactions)
                 sets_init, self.dict_sets_support = self.filterSet(set_trans, inititems)
                 setlen = 2
                 items_tmp = self.combine_lists(sets_init, setlen)
                 self.freq_sets, sup_tmp = self.filterSet(set_trans, items_tmp)
                 self.dict_sets_support.update(sup_tmp)
                 self.ass_matrix = np.zeros((nitems,nitems))
                 for freqset in self.freq_sets:
                     #print 'freqset',freqset
                     list_setitems = [frozenset([item]) for item in freqset]
                     #print "freqSet", freqset, 'H1', list_setitems
                     self.calc_confidence_matrix(freqset, list_setitems)

             def filterSet(self,set_trans, likeditems):
                 itemscnt = {}
                 for id in set_trans:
                     for item in likeditems:
                         if item.issubset(id):
                             itemscnt.setdefault(item, 0)
                             itemscnt[item] += 1
                 num_items = float(len(set_trans))
                 freq_sets = []
                 dict_sets = {}
                 for key in itemscnt:
                     support = itemscnt[key] / num_items
```

```
            if support >= self.min_support:
                freq_sets.insert(0, key)
            dict_sets[key] = support
        return freq_sets, dict_sets

    def combine_lists(self,freq_sets, setlen):
        setitems_list = []
        nsets = len(freq_sets)
        for i in range(nsets):
            for j in range(i + 1, nsets):
                setlist1 = list(freq_sets[i])[:setlen - 2]
                setlist2 = list(freq_sets[j])[:setlen - 2]
                if set(setlist1) == set(setlist2):
                    setitems_list.append(freq_sets[i].union(freq_sets[j]))
        return setitems_list

    def calc_confidence_matrix(self,freqset, list_setitems):
        for target in list_setitems:
            confidence = self.dict_sets_support[freqset] / self.dict_sets_support[freqset - target]
            if confidence >= self.min_confidence:
                self.ass_matrix[list(freqset - target)[0]][list(target)[0]] = confidence

    def GetRecItems(self,u_vec,indxs=False):
        vec_recs = np.dot(u_vec,self.ass_matrix)
        sortedweight = np.argsort(vec_recs)
        seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
        seenmovies = np.array(self.Movieslist)[seenindxs]
        #remove seen items
        recitems = np.array(self.Movieslist)[sortedweight]
        recitems = [m for m in recitems if m not in seenmovies]
        if indxs:
            vec_recs[seenindxs]=-1
            recsvec = np.argsort(vec_recs)[::-1][np.argsort(vec_recs)>0]
            return recsvec
        return recitems[::-1]
```

类构造器的参数有效用矩阵 Umatrix、电影名称列表 Movieslist、支持度阈值 min_support（默认为 0.1）、置信度阈值 min_confidence（默认为 0.1）和将交易数据中电影纳入考虑范围的最低分数 likethreshold（默认为 3）。函数 combine_lists 寻找所有可能的规则，而 filterSet 过滤规则，找出满足最小支持度阈值的规则。calc_confidence_matrix 用满足最小阈值（否则默认为 0）的置信度填充 ass_matrix 矩阵。GetRecItems 根据用户的分数 u_vec，返回电影推荐列表。

5.6 对数似然比推荐方法

对数似然比（Log-Likelihood Ratio，LLR）是度量事件 A、B 不太可能是独立事件，而是共现几率大于偶然（比一个事件单独发生更加频繁）的一种方法。换言之，LLR 表明事件 A 和 B 共现的频率，比正常分布（两个事件上的分布）所能预测的要显著。

Ted Dunning (http://tdunning.blogspot.it/2008/03/surprise-and-coincidence.html) 讲过，我们可以借助事件 A 和 B 的二项分布来定义 LLR，二项分布矩阵 k 及 LLR 的公式为：

表 5.1

	A	Not A
B	k11	k12
Not B	k21	k22

$$LLR = 2N\left(H\left(\left[k_{11,}k_{12,}k_{21,}k_{22,}\right]\right) - H\left(\left[k_{11,}k_{12,}k_{21,}k_{22,}\right]\right) - \left(\left[k_{11,}k_{12,}k_{21,}k_{22,}\right]\right)\right)$$

其中，$N = k_{11} + k_{12} + k_{21} + k_{22}, H(p) = \sum_{i=0}^{len(p)} p_i / N \log(p_i / N)$ 为度量向量 p 所包含信息的**香农熵**。

注意：$H([k_{11}, k_{12}, k_{21}, k_{22}]) - H([k_{11} + k_{12} + k_{21} + k_{22}]) - H([k_{11} + k_{21} + k_{12} + k_{22}])$ 也称为两个事件变量 A 和 B 的**互信息**（Mutual Information，MI），它度量的是两个事件的发生是如何彼此相关的。

这种检验方法也叫作 $G2$，已证实它在检测稀少事件（尤其是文本分析）的共现方面非常有效，因此可用于数据稀疏的情况（比如例子中的效用矩阵）。

再回到电影推荐的例子，事件 A 和 B 对应的是用户喜欢或不喜欢电影 A 和 B。事件“喜欢一部电影”的定义是，打分大于 3 分（反之，不喜欢）。我们用下面这个类实现该算法：

```
In [19]: class LogLikelihood(object):
             def __init__(self,Umatrix,Movieslist,likethreshold=3):
                 self.Movieslist = Movieslist
                 #calculate loglikelihood ratio for each pair
                 self.nusers = len(Umatrix)
                 self.Umatrix =Umatrix
                 self.likethreshold = likethreshold
                 self.likerange = range(self.likethreshold+1,5+1)
                 self.dislikerange = range(1,self.likethreshold+1)
                 self.loglikelihood_ratio()

             def calc_k(self,a,b):
                 tmpk = [[0 for j in range(2)] for i in range(2)]
                 for ratings in self.Umatrix:
                     if ratings[a] in self.likerange and ratings[b] in self.likerange:
                        tmpk[0][0] += 1
                     if ratings[a] in self.likerange and ratings[b] in self.dislikerange:
                         tmpk[0][1] += 1
                     if ratings[a] in self.dislikerange and ratings[b] in self.likerange:
                         tmpk[1][0] += 1
                     if ratings[a] in self.dislikerange and ratings[b] in self.dislikerange:
                         tmpk[1][1] += 1
                 return tmpk

             def calc_llr(self,k_matrix):
                 Hcols=Hrows=Htot=0.0
                 if sum(k_matrix[0])+sum(k_matrix[1])==0:
                     return 0.
                 invN = 1.0/(sum(k_matrix[0])+sum(k_matrix[1]))
                 for i in range(0,2):
                     if((k_matrix[0][i]+k_matrix[1][i])!=0.0):
                        Hcols += invN*(k_matrix[0][i]+k_matrix[1][i])*math.log((k_matrix[0][i]+k_matrix[1][i])*invN )#sum of row
                     if((k_matrix[i][0]+k_matrix[i][1])!=0.0):
                        Hrows += invN*(k_matrix[i][0]+k_matrix[i][1])*math.log((k_matrix[i][0]+k_matrix[i][1])*invN )#sum of col
                     for j in range(0,2):
                         if(k_matrix[i][j]!=0.0):
                            Htot +=invN*k_matrix[i][j]*math.log(invN*k_matrix[i][j])
                 return 2.0*(Htot-Hcols-Hrows)/invN

             def loglikelihood_ratio(self):
                 nitems = len(self.Movieslist)
                 self.items_llr= pd.DataFrame(np.zeros((nitems,nitems))).astype(float)
                 for i in xrange(nitems):
                     for j in xrange(nitems):
                         if(j>=i):
                            tmpk=self.calc_k(i,j)
                            self.items_llr.ix[i,j] = self.calc_llr(tmpk)
                         else:
                            self.items_llr.ix[i,j] = self.items_llr.iat[j,i]

             def GetRecItems(self,u_vec,indxs=False):
                 items_weight = np.dot(u_vec,self.items_llr)
                 sortedweight = np.argsort(items_weight)
                 seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
                 seenmovies = np.array(self.Movieslist)[seenindxs]
                 #remove seen items
                 recitems = np.array(self.Movieslist)[sortedweight]
                 recitems = [m for m in recitems if m not in seenmovies]
                 if indxs:
                     items_weight[seenindxs]=-1
                     recsvec = np.argsort(items_weight)[::-1][np.argsort(items_weight)>0]
                     return recsvec
                 return recitems[::-1]
```

构造器的参数有效用矩阵、电影名称列表和判断用户是否喜欢电影的阈值 likethreshold（默认为 3）。函数 loglikelihood_ratio 计算矩阵 k(calc_k)和相应的 LLR (calc_llr)，为每一对电影 i 和 j 生成 LLR 值。函数 GetRecItems 返回用户（该用户的打分数据为 u_vec）的电影推荐列表（该方法不预测打分数据）。

5.7 混合推荐系统

这类方法为了达到更理想的效果，在推荐引擎中结合使用 CBF 和 CF 方法。业界尝试过多种混合推荐方法，归结起来不外乎以下几种：

- **加权**：对 CBF 和 CF 预测得到的分数，取加权平均值。
- **混合**：分别用 CF 和 CBF 预测，然后将预测结果合并为一个列表。
- **切换**：根据特定的规则，选择使用 CF 或 CBF 预测方法。
- **特征组合**：综合考虑 CF 和 CBF 的特征，找到最相似的用户或商品。
- **特征扩充**（Feature augmentation）：类似于特征组合，但还要用附加特征预测分数，然后主推荐引擎使用得到的分数生成推荐列表。例如，对于基于内容的模型未打过分的电影，内容增强型协同过滤（Content-Boosted Collaborative Filtering）学习为其打分，然后再用协同过滤方法确定推荐列表。

举个例子，我们结合基于商品特征的 CBF 方法和基于用户的 CF 方法，实现两种混合的特征组合方法。方法一采用基于用户的 CF，扩展效用矩阵，增加每种电影类型每位用户的平均分。具体实现方法见下面这个类：

```
In [37]: class Hybrid_cbf_cf(object):
             def __init__(self,Movies,Movieslist,Umatrix):
                 #calc user profiles:
                 self.nfeatures = len(Movies[0])
                 self.Movieslist = Movieslist
                 self.Movies = Movies.astype(float)
                 self.Umatrix_mfeats = np.zeros((len(Umatrix),len(Umatrix[0])+self.nfeatures))
                 means = np.array([ Umatrix[i][Umatrix[i]>0].mean() for i in xrange(len(Umatrix))]).reshape(-1,1)
                 diffs = np.array([ [Umatrix[i][j]-means[i] if Umatrix[i][j]>0 else 0.
                                     for j in xrange(len(Umatrix[i]))  ] for i in xrange(len(Umatrix))])
                 self.Umatrix_mfeats[:,:len(Umatrix[0])] = Umatrix#diffs
                 self.nmovies = len(Movies)
                 #calc item features for each user
                 for u in xrange(len(Umatrix)):
                     u_vec = Umatrix[u]
                     self.Umatrix_mfeats[u,len(Umatrix[0]):] = self.GetUserItemFeatures(u_vec)

             def GetUserItemFeatures(self,u_vec):
                 mean_u = u_vec[u_vec>0].mean()
                 #diff_u = u_vec-mean_u
                 features_u = np.zeros(self.nfeatures).astype(float)
                 cnts = np.zeros(self.nfeatures)
                 for m in xrange(self.nmovies):
                     if u_vec[m]>0:#u has rated m
                         features_u += self.Movies[m]*u_vec[m]#self.Movies[m]*(diff_u[m])
```

```
                cnts += self.Movies[m]
        #average:
        for m in xrange(self.nfeatures):
            if cnts[m]>0:
                features_u[m] = features_u[m]/float(cnts[m])
        return features_u

        def CalcRating(u_vec,r,neighs):
            rating = 0.
            den = 0.
            for j in xrange(len(neighs)):
                rating += neighs[j][-1]*float(neighs[j][r]-neighs[j][neighs[j]>0][:-1].mean())
                den += abs(neighs[j][-1])
            if den>0:
                rating = np.round(u_vec[u_vec>0].mean()+(rating/den),0)
            else:
                rating = np.round(u_vec[u_vec>0].mean(),0)
            if rating>5:
                return 5.
            elif rating<1:
                return 1.
            return rating
        #add similarity col
        nrows = len(self.Umatrix_mfeats)
        ncols = len(self.Umatrix_mfeats[0])
        data_sim = np.zeros((nrows,ncols+1))
        data_sim[:,:-1] = self.Umatrix_mfeats
        u_rec = np.zeros(len(u_vec))
        #calc similarities:
        mean = u_vec[u_vec>0].mean()
        u_vec_feats = u_vec#np.array([u_vec[i]-mean if u_vec[i]>0 else 0 for i in xrange(len(u_vec))])
        u_vec_feats = np.append(u_vec_feats,self.GetUserItemFeatures(u_vec))

        for u in xrange(nrows):
            if np.array_equal(data_sim[u,:-1],u_vec)==False: #list(data_sim[u,:-1]) != list(u_vec):
                data_sim[u,ncols] = sim(data_sim[u,:-1],u_vec_feats)
            else:
                data_sim[u,ncols] = 0.
        #order by similarity:
        data_sim[:,:-1] = self.Umatrix_mfeats
        u_rec = np.zeros(len(u_vec))
        #calc similarities:
        mean = u_vec[u_vec>0].mean()
        u_vec_feats = u_vec#np.array([u_vec[i]-mean if u_vec[i]>0 else 0 for i in xrange(len(u_vec))])
        u_vec_feats = np.append(u_vec_feats,self.GetUserItemFeatures(u_vec))

        for u in xrange(nrows):
            if np.array_equal(data_sim[u,:-1],u_vec)==False: #list(data_sim[u,:-1]) != list(u_vec):
                data_sim[u,ncols] = sim(data_sim[u,:-1],u_vec_feats)
            else:
                data_sim[u,ncols] = 0.
        #order by similarity:
        data_sim =data_sim[data_sim[:,ncols].argsort()][::-1]
        #find the K users for each item not rated:

        for r in xrange(self.nmovies):
            if u_vec[r]==0:
                neighs = FindKNeighbours(r,data_sim,K)
                #calc the predicted rating
                u_rec[r] = CalcRating(u_vec,r,neighs)
        return u_rec
```

构造器生成新的效用矩阵 Umatrix_mfeats，为每位用户增加了他为每种类型的电影打的平均分这个特征。函数 CalcRatings 使用皮尔逊相关系数度量方法，比较每位用户扩展后的特征向量，找出 K 个近邻。方法二，用 SVD 方法扩展效用矩阵，将每位用户的类型喜好整合进去。

```
In [22]: class Hybrid_svd(object):
             def __init__(self,Movies,Movieslist,Umatrix,K,inp):
                 #calc user profiles:
                 self.nfeatures = len(Movies[0])
                 self.Movieslist = Movieslist
                 self.Movies = Movies.astype(float)
```

```
        R_tmp = copy.copy(Umatrix)
        R_tmp = R_tmp.astype(float)
        #imputation

        if inp != 'none':
            R_tmp = imputation(inp,Umatrix)
        Umatrix_mfeats = np.zeros((len(Umatrix),len(Umatrix[0])+self.nfeatures))
        means = np.array([ Umatrix[i][Umatrix[i]>0].mean() for i in xrange(len(Umatrix))]).reshape(-1,1)
        diffs = np.array([ [float(Umatrix[i][j]-means[i])
                          if Umatrix[i][j]>0 else float(R_tmp[i][j]-means[i]) for j in xrange(len(Umatrix[i])) ]
                      for i in xrange(len(Umatrix))])
        Umatrix_mfeats[:,:len(Umatrix[0])] = diffs#R_tmp
        self.nmovies = len(Movies)
        #calc item features for each user
        for u in xrange(len(Umatrix)):
            u_vec = Umatrix[u]
            Umatrix_mfeats[u,len(Umatrix[0]):] = self.GetUserItemFeatures(u_vec)

        #calc svd
        svd = TruncatedSVD(n_components=K, random_state=4)
        R_k = svd.fit_transform(Umatrix_mfeats)
        R_tmp = means+svd.inverse_transform(R_k)
        self.matrix = np.round(R_tmp[:,:self.nmovies],0)


    def GetUserItemFeatures(self,u_vec):
        mean_u = u_vec[u_vec>0].mean()
        diff_u = u_vec-mean_u
        features_u = np.zeros(self.nfeatures).astype(float)
        cnts = np.zeros(self.nfeatures)
        for m in xrange(self.nmovies):
            if u_vec[m]>0:#u has rated m
               features_u += self.Movies[m]*(diff_u[m])#self.Movies[m]*u_vec[m]
               cnts += self.Movies[m]
        #average:
        for m in xrange(self.nfeatures):
            if cnts[m]>0:
               features_u[m] = features_u[m]/float(cnts[m])
        return features_u
```

SVD 方法，每位用户所打的分数减去了该用户的平均分。每位用户的类型偏好分数减去了他打的平均分。

5.8 推荐系统评估

我们讨论了与当今商业应用联系最为密切的方法，现在我们再来学习推荐系统的评估方法。评估可在线下（只用效用矩阵的数据）或线上（用效用矩阵和网站用户实时产生的数据）进行。第 7 章将结合线上电影推荐系统讲解线上评估方法。这一节，我们采用评估推荐系统常用的两种线下检验方法，来评估这些推荐方法的性能：分数的均方根误差（Root Mean Square Error，RMSE）和排序的正确率。所有评估方法，均采用 k-折交叉检验（第 3 章），为了保证结果的客观性，我们采用 5 折交叉检验。用以下函数将效用矩阵分为 5 折：

```
In [42]: def cross_validation(df,k):
             val_num = int(len(df)/float(k))
             print val_num
             df_trains = []
             df_vals = []
             for i in xrange(k):
                 start_val = (k-i-1)*val_num
                 end_val = start_val+val_num
                 df_trains.append(pd.concat([df[:start_val],df[end_val:]]))
                 df_vals.append(df[start_val:end_val])

             return df_trains,df_vals
```

DataFrame 对象 df 存储效用矩阵，*k* 为折数。验证集中每位用户的分数向量 u_vec，我们随机隐藏了半数电影的分数，以便预测它们的实际值。

```
In [23]: import random
         def HideRandomRatings(u_vec, ratiovals=0.5):
             u_test = np.zeros(len(u_vec))
             u_vals = np.zeros(len(u_vec))
             cnt = 0
             nratings = len(u_vec[u_vec>0])
             for i in xrange(len(u_vec)):
                 if u_vec[i]>0:
                     if bool(random.getrandbits(1)) or cnt>=int(nratings*ratiovals):
                         u_test[i]=u_vec[i]
                     else:#random choice to hide the rating:
                         cnt +=1
                         u_vals[i]=u_vec[i]
             return u_test,u_vals
```

u_vals 存储预测值，u_test 存储用于测试算法的实际分数。比较不同算法不同度量方法之前，我们加载效用矩阵、电影内容矩阵到 DataFrame 对象，将数据分成 5 折，使用交叉检验方法。

```
In [24]: #load data
         df = pd.read_csv('data/utilitymatrix.csv')
         print df.head(4)
         df_movies = pd.read_csv('data/movies_content.csv')
         movies = df_movies.values[:,1:]
         print 'check:::',len(df.columns[1:]),'--',len(df_movies)
         movieslist = list(df.columns[1:])
         #k-fold cv 5 folds
         nfolds = 5
         df_trains,df_vals = cross_validation(df,nfolds)
```

验证集包含在 df_vals 中，我们需要用本节的 HideRandomRatings 函数隐藏用户实际打的分数。

```
In [31]: nmovies = len(df_vals[0].values[:,1:][0])
         s = []
         tests_vecs_folds = []
         for i in xrange(nfolds):
             u_vecs = df_vals[i].values[:,1:]
             vtests = np.empty((0,nmovies),float)
             vvals = np.empty((0,nmovies),float)
             for u_vec in u_vecs:
                 u_test,u_vals = HideRandomRatings(u_vec)
                 vvals = np.vstack([vvals,u_vals])
                 vtests = np.vstack([vtests,u_test])
             vals_vecs_folds.append(vvals)
             tests_vecs_folds.append(vtests)
```

矩阵 movies、movieslist 列表以及 DataFrame 对象 df_trains、vals_vecs_folds 和 tests_vecs_folds 都已准备就绪，接下来我们训练和验证前面几节讨论的各种方法。我们首先评估它们的**均方根误差**（RMSE）。

5.8.1 均方根误差（RMSE）评估

均方根误差检验方法只适用于 CF 和线性回归 CBF，因为只有这些算法才生成预测分数。

给定验证集 u_vals 的每个分数 r_{ij}，用每种方法计算得到预测分数 $\hat{r}_{ij}$，相应的均方根误差为：

$$\text{RMSE}=\sqrt{\frac{\sum_{i,j\in u_vals}(r_{ij}-\hat{r}_{ij})^2}{N_{val}}}$$

其中，N_{val} 为 u_vals 向量中打分数据的数量。公式中的平方因子严厉惩罚误差较大的情况，因此 RMSE（最佳分数）较低的方法，它们的误差较小，且分散于对所有分数的预测，而不是少数分数误差较大，而不像平均绝对误差 $\text{MAE}=\sqrt{\frac{\sum_{i,j\in u_vals}(r_{ij}-\hat{r}_{ij})}{N_{val}}}$ 所刻画的那样。

基于记忆的 CF、基于用户和商品的协同过滤方法，它们的 RMSE 计算方法见下面代码：

```
In [43]: def SE(u_preds,u_vals):
             nratings = len(u_vals)
             se = 0.
             cnt = 0
             for i in xrange(nratings):
                 if u_vals[i]>0:
                     se +=  (u_vals[i]-u_preds[i])*(u_vals[i]-u_preds[i])
                     cnt += 1
             return se,cnt
```

```
In [40]: err_itembased = 0.
         cnt_itembased = 0
         err_userbased = 0.
         cnt_userbased = 0
         err_slopeone = 0.
         cnt_slopeone = 0
         err_cbfcf = 0.
         cnt_cbfcf = 0
         for i in xrange(nfolds):
             Umatrix = df_trains[i].values[:,1:]
             cfitembased = CF_itembased(Umatrix)
             cfslopeone = SlopeOne(Umatrix)
             cbfcf = Hybrid_cbf_cf(movies,movieslist,Umatrix)
             print 'fold:',i+1
             vec_vals = vals_vecs_folds[i]
             vec_tests = tests_vecs_folds[i]
             for j in xrange(len(vec_vals)):
                 u_vals = vec_vals[j]
                 u_test = vec_tests[j]
                 #cbfcf
                 u_preds = cbfcf.CalcRatings(u_test,5)
                 e,c =  SE(u_preds,u_vals)
                 err_cbfcf +=e
                 cnt_cbfcf +=c
                 #cf_userbased
                 u_preds = CF_userbased(u_test,5,Umatrix)
                 e,c =  SE(u_preds,u_vals)
                 err_userbased +=e
                 cnt_userbased +=c
                 #cf_itembased
                 u_preds = cfitembased.CalcRatings(u_test,5)
                 e,c =  SE(u_preds,u_vals)
                 err_itembased +=e
                 cnt_itembased +=c
                 #slope one
                 u_preds = cfslopeone.CalcRatings(u_test,5)
                 e,c =  SE(u_preds,u_vals)
                 err_slopeone +=e
                 cnt_slopeone +=c
         rmse_userbased = np.sqrt(err_userbased/float(cnt_userbased))
         rmse_itembased = np.sqrt(err_itembased/float(cnt_itembased))
         rmse_slopeone = np.sqrt(err_slopeone/float(cnt_slopeone))
         print 'user_userbased rmse:',rmse_userbased,'--',cnt_userbased
         print 'user_itembased rmse:',rmse_itembased,'--',cnt_itembased
         print 'slope one rmse:',rmse_slopeone,'--',cnt_slopeone

         rmse_cbfcf = np.sqrt(err_cbfcf/float(cnt_cbfcf))
         print 'cbfcf rmse:',rmse_cbfcf,'---',cnt_cbfcf
```

对于每种方法，调用 SE 函数，计算每一折的误差，然后得到所有折的误差。

基于商品的 CF 和 slope one 算法使用 5 个近邻，基于用户的 CF 使用 20 个近邻，各方法的误差见表 5.2。

表 5.2

方法	RMSE	预测分数的数量
基于用户的 CF	1.01	39972
基于商品的 CF	1.03	39972
Slope one	1.08	39972
基于用户的 CF-CBF	1.01	39972

以上方法，与 RMSE 类似，但最佳方法是基于商品的协同过滤。

对于基于模型的方法，我们不再隐藏验证集分数，而是将 u_test 整合到效用矩阵一起训练，然后用下面代码计算 RMSE：

```
In [63]: err_svd = 0.
cnt_svd = 0
err_svd_em = 0.
cnt_svd_em = 0
err_als = 0.
cnt_als = 0
err_cbfreg = 0.
cnt_cbfreg = 0
for i in xrange(nfolds):
    Umatrix = df_trains[i].values[:,1:]
    print 'fold:',i+1
    teststartindx = len(Umatrix)
    vals_vecs = vals_vecs_folds[i]
    tests_vecs = tests_vecs_folds[i]
    for k in xrange(len(vals_vecs)):
        u_vals = vals_vecs[k]
        u_test = tests_vecs[k]
        #add test vector to utility matrix
        Umatrix = np.vstack([Umatrix,u_test])

    #svd_em_matrix = Hybrid_svd(movies,movieslist,Umatrix,20,'useraverage').matrix#SVD_EM(Umatrix,20,'useraverage',1)
    svd_matrix = SVD(Umatrix,20,'itemaverage')
    cbf_reg = CBF_regression(movies,Umatrix)
    #als_umatrix = SGD(Umatrix,20,50)#ALS(Umatrix,20,50)#NMF_alg(Umatrix,20,'itemaverage',0.001)
    #evaluate errors
    for indx in xrange(len(vals_vecs)):
        #e,c =  SE(als_umatrix[teststartindx+indx],vals_vecs[indx])
        #err_als += e
        #cnt_als += c
        u_preds = cbf_reg.CalcRatings(Umatrix[teststartindx+indx])
        e,c = SE(u_preds,vals_vecs[indx])
        err_cbfreg +=e
        cnt_cbfreg +=c

        e,c = SE(svd_matrix[teststartindx+indx],vals_vecs[indx])
        err_svd +=e
        cnt_svd +=c
        #e,c = SE(svd_em_matrix[teststartindx+indx],vals_vecs[indx])
        #err_svd_em +=e
        #cnt_svd_em +=c

if cnt_svd==0: cnt_svd=1
if cnt_svd_em==0: cnt_svd_em=1
if cnt_als==0: cnt_als=1
if cnt_cbfreg==0: cnt_cbfreg=1

rmse_als = np.sqrt(err_als/float(cnt_als))
rmse_svd = np.sqrt(err_svd/float(cnt_svd))
rmse_svd_em = np.sqrt(err_svd_em/float(cnt_svd_em))
rmse_cbfreg = np.sqrt(err_cbfreg/float(cnt_cbfreg))

print 'svd rmse:',rmse_svd,'--',cnt_svd
#print 'svd_em rmse:',rmse_svd_em,'--',cnt_svd_em
#print 'als rmse:',rmse_als,'--',cnt_als
print 'cbfreg rmse:',rmse_cbfreg,'--',cnt_cbfreg
```

上述代码只计算了 CBF 回归和 SVD 方法的 RMSE，读者可以利用这段代码计算其他算法的误差，因为所需要的大多数代码都以注释的形式添加到上述代码中（最大期望 SVD、SGD、ALS 和 NMF）。结果见表 5.3（K 为特征空间的维度）：

表 5.3

方法	RMSE	预测分数的数量
CBF 线性回归（α= 0.01、l =0.0001、50 次迭代）	1.09	39972
SGD（K=20、50 次迭代、a =0.00001、l=0.001）	1.35	39972
ALS（K=20、50 次迭代、l =0.001）	2.58	39972
SVD（插值=用户平均分数、K =20）	1.02	39972
SVD EM（插值=商品平均分数、30 次迭代、K =20）	1.03	39972
混合 SVD（插值=用户平均分数、K =20）	1.01	39972
NMF（K =20、插值=用户平均分数）	0.97	39972

不出所料，ALS 和 SGD 表现最差，但是用它们讲解推荐系统有助于理解，因此我们对它们予以讨论（还有一个原因，它们的实现没有 sklearn 库所实现的几种方法那么完善）。

其他方法结果相差不大。然而，需注意的是，混合方法比起单独的 SVD、基于用户的 CF 算法只好一点点。还要注意的是，我们随机选择要预测的电影，因此多次比较，其结果可能不同。

5.8.2 分类效果的度量方法

打分误差 RMSE 不能实际表明各方法性能的好坏，只是一种学术评价方法，在实际商业环境是不会使用的。网站的目标是向用户推荐相关内容，而不管用户的实际打分。我们用准确率、召回率和 f_1 值（见第 3 章）评估推荐商品的相关性。预测结果中正确的是指分数大于 3 的电影。我们计算每种算法返回的前 50 部电影的这 3 个值（如果算法返回电影列表，若不返回则取 50 个最高的预测分数）。这些指标的计算方式如下：

```
In [33]: def ClassificationMetrics(vec_vals,vec_recs,likethreshold=3,shortlist=50,ratingsval=False,vec_test=None):
             #convert vals in indxs vec
             indxs_like = [i for i in xrange(len(vec_vals)) if vec_vals[i]>likethreshold]
             indxs_dislike = [i for i in xrange(len(vec_vals)) if vec_vals[i]<=likethreshold and vec_vals[i]>0]
             cnt = len(indxs_like)+len(indxs_dislike)
             indxs_rec = []
             if ratingsval:
                 #convert ratings into items's list
                 if vec_test==None:
                     raise 'Error no test vector'
                 indxs_rec = [i for i in xrange(len(vec_recs)) if vec_recs[i]>likethreshold and vec_test[i]<1][:shortlist]
             else:
                 #consider only the first slot of recs
                 indxs_rec = vec_recs[:shortlist]

             tp = len(set(indxs_rec).intersection(set(indxs_like)))
             fp = len(set(indxs_rec).intersection(set(indxs_dislike)))
             fn = len(set(indxs_like)^(set(indxs_rec).intersection(set(indxs_like))))
             precision = 0.
             if tp+fp>0:
                 precision = float(tp)/(tp+fp)
             recall = 0.
             if tp+fn>0:
                 recall = float(tp)/(tp+fn)
             f1 = 0.
             if recall+precision >0:
                 f1 = 2.*precision*recall/(precision+recall)

             return np.array([precision,recall,f1]),cnt
```

上述代码，布尔型变量 ratingsval 表示推荐方法返回的是分数还是电影列表。我们用函数 ClassificationMetrics 像计算所有方法的 RMSE 那样计算各指标，计算各指标的代码没有列在这里（你可以练习自己编写）。表 5.4 总结了各种方法的 3 个指标（*neighs* 为近邻数、*K* 为特征空间的维度）：

表 5.4

方法	准确率	召回率	*f*1	预测分数的数量
基于用户的 CF (neighs =20)	0.6	0.18	0.26	39786
基于用户的 CBFCF (neighs =20)	0.6	0.18	0.26	39786
混合 SVD (*K* =20、插值=用户平均分数)	0.54	0.12	0.18	39786
基于商品的 CF (neighs =5)	0.57	0.15	0.22	39786
Slope one (neighs =5)	0.57	0.17	0.24	39,786
SVD EM (*K* =20、30 次迭代、插值=用户平均分数)	0.58	0.16	0.24	39786
SVD (*K* =20、插值=商品平均分数)	0.53	0.12	0.18	39786
CBF 回归 (α= 0.01、l =0.0001、50 次迭代)	0.54	0.13	0.2	39786
SGD (*K*=20、α=0.00001、l =0.001)	0.52	0.12	0.18	39786
ALS (*K* =20、λ =0.001、50 次迭代)	0.57	0.15	0.23	39,786
CBF 平均	0.56	0.12	0.19	39786
LLR	0.63	0.3	0.39	39786
NMF (*K* =20、λ =0.001、插值方法为 ssss)	0.53	0.13	0.19	39786
关联规则	0.68	0.31	0.4	39786

由表 5.4 可见，最佳推荐方法是关联规则，LLR、基于用户的混合 CBFCF、基于用户的 CF 准确率较高。再次提醒，由于每次是随机选择电影进行预测，预测结果也会有所不同。

5.9 小结

本章讨论了最常用的推荐系统方法：协同过滤、基于内容的过滤和两种简单的混合算法。我们查询相关文献时，可能会遇到模态推荐系统（modal recommendation），它是将多种不同的数据（用户性别、人口统计学特征、观点、地理位置、设备等）整合到算法之中。这些更为高级的方法需要用多种不同的数据。

在第 7 章中，我们将用本章讨论的方法实现一个 Web 推荐系统。在这之前，我们先用第 6 章一章的篇幅来介绍搭建 Web 应用所使用的 Django 框架。

第 6 章
开始 Django 之旅

开源 Web 框架 Django 简单易用，稳定性和灵活性高，因此被广泛应用于商业环境（它充分利用了 Python 拥有丰富的库这一优势）。

我们可以用 Web 应用来管理和分析数据，开发 Web 应用要用到 Web 框架的相关功能，本章重点讲解 Django 框架的这些功能。此外，我们还会解释搭建完整的 Web 应用包括哪些主要环节，但更多细节和信息限于篇幅，不再赘述，请自行查阅官方文档 https://docs.djangoproject.com 或其他资料。我们将介绍 Web 服务器应用的主要概念（配置、模型和命令）、HTML 和 shell 的基础知识、REST 框架接口的主要概念及在 Django 中它们是如何实现的（serializer、REST 调用和 swagger）。我们会简要介绍如何用 HTTP GET、POST 方法在因特网上传输数据，还会讲解 Django 的安装方法以及如何用它搭建 Web 服务器。

6.1 HTTP——GET 和 POST 方法的基础

超文本传输协议（Hypertext Transfer Protocol，HTTP）实现了客户端（比如 Web 浏览器）和服务器（我们的应用）之间的交互。给定网页的 URL 地址，客户端使用 GET 方法向服务器查询数据，查询词在 URL 中是以参数形式指定的。若用 curl 命令来解释，如下所示：

```
curl -X GET url_path?name1=value1&name2=value2
```

URL?号符号后面的键值对，指定的是要查询的数据，多项数据之间用&符号分隔。

客户端将数据传送给服务器的方法叫作 POST。POST 方法将被传输的数据放到请求的 *body* 部分：

```
curl -X POST -d @datafile.txt url_path
```

现在，我们开始讨论如何用 Django 搭建 Web 服务器和 Web 应用。

6.1.1　Django 的安装和服务器的搭建

在终端输入以下命令，安装 Django 库：

```
sudo pip instal django
```

该命令应该安装的是 1.7 或以上版本的 Django（作者用的是 1.7 版本）。新建 Web 应用，请用以下命令：

```
django-admin startproject test_server
```

上述命令生成 test_app[①]新文件夹，该文件夹的文件目录树如下：

```
└── test_server
    ├── manage.py
    └── test_server
        ├── __init__.py
        ├── settings.py
        ├── urls.py
        └── wsgi.py
```

test_server 文件夹中有 manage.py 文件，开发人员可用它执行多种操作。该文件夹下，还有一个同名的子文件夹 test_server，它里面有以下文件：

- settings.py：存储服务器的所有参数设置；
- urls.py：汇总 Web 应用的所有 URL 路径及对应的函数名，这些函数位于 views.py 文件中，其作用是渲染模板，生成 URL 所指向的网页；
- wsgi.py：与 Web 应用进行服务器通信的模块；
- __init__.py：将每个文件夹定义为一个包，以便从包内导入模块。

输入以下命令，在我们本地的机器上，访问 http://127.0.0.1:8080/，可看到 Django 的欢迎页（Welcome to Django）：[②]

```
python manage.py runserver 8080
```

① 应为 test_server。——译者注

② 用到 manage.py 的命令，需到 manage.py 所在的目录下执行，请到命令行工具中切换到该目录。——译者注

其中，8080 为服务器开启的端口（如不指定，默认开启 8000 端口）。现在服务器已经就绪，我们可以用如下命令创建 Web 应用，想创建多少就能创建多少。

```
python manage.py startapp nameapp
```

上述命令在 test_app[①]文件夹的根目录下，新建一个文件夹 nameapp：

```
├── manage.py
├── nameapp
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── test_server
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

我们先解释最重要的参数配置，之后再讨论该文件夹中的内容及其功能。请注意，Django 1.9 版本，nameapp 文件夹包含 apps.py 文件，它改用 apps.py 文件配置 nameapp 应用，不再用 settings.py 文件。

6.1.2 配置

settings.py 文件存储 Django 服务器运行所需的全部配置。需要设置的、最重要的参数如下所示。

（1）除了默认安装的、网站管理所需的一般性 Django 应用，我们还要安装 REST 框架：

```
INSTALLED_APPS = (
...
'rest_framework',
'rest_framework_swagger',
'nameapp',
)
```

① 应为 test_server。——译者注

安装了 REST 框架，Django 应用（例子中的 nameapp）就可以用 REST API 进行通信，REST Framework Swagger 是一种管理 REST API 的 Web 交互接口。后续章节会讨论这些功能。还要注意，我们创建的每一种应用（例子中的 nameapp）都要添加到 INSTALLED_APPS 字段中。

（2）Diango 支持用多种后端数据库（MySQL、Oracle 和 PostgreSQL 等）存储数据。该例使用 SQLite3（默认选项）：

```
DATABASES = {
       'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'mydatabase',
        }
       }
```

网页用 HTML 编码实现，因此需要专门准备一个文件夹，存储 HTML 代码。我们习惯用 templates 文件夹存储网页布局文件：

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)
```

（3）修饰网站的 CSS 样式表和 JavaScript 代码通常单独存储到 static 文件夹，将它放到 test_server 文件夹中。要获取该文件夹下的文件，需要进行如下配置：

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'static')
STATIC_URL = '/static/'
MEDIA_URL = ''
STATIC_ROOT = ''
STATICFILES_DIRS = ( os.path.join(BASE_DIR, "static"), )
```

（4）网站的 URL 设置，需做以下配置，URL 将从该文件中获取（该例中的 test_server/urls.py 文件）：

```
ROOT_URLCONF = 'test_server.urls'
```

（5）我们可以新建一文件，存储为解决 bug 而输出的语句，放到这个文件中。我们使用 logging 库，配置方式如下：

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
```

```
        'formatters': {
            'standard': {
                'format': '%(asctime)s %(levelname)s %(name)s
    %(message)s'
            },
        },
        'handlers': {
            'default': {
                'level':'DEBUG',
                'class':'logging.handlers.RotatingFileHandler',
                'filename': 'test_server.log',
                'maxBytes': 1024*1024*5, # 5 MB
                'backupCount': 5,
                'formatter':'standard',
            },
        },
    'loggers': {
            '': {
                'handlers': ['default'],
                'level': 'DEBUG',
                'propagate': True
            },
        }
    }
```

做好上述配置，我们用 logging 库定义的所有输出语句，将存储到 test_server.log 文件（例如，logging.debug('write something')）。

至此，settings.py 里所有重要的项，均已配置完毕。接下来，我们可以集中精力开发一个简单的邮件地址簿应用。按照前面讲的方法，新建一个应用：

```
python manage.py startapp addresesapp
```

接着，在服务器（addressapp）文件夹根目录下的 test_server 文件夹中，新建页面模板文件夹 templates 和静态内容文件夹 static：

```
├── addresesapp
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   ├── models.py
│   ├── tests.py
│   └── views.py
```

```
├── manage.py
└── test_server
├── __init__.py
├── __init__.pyc
├── settings.py
├── settings.pyc
├── static
├── templates
├── urls.py
└── wsgi.py
```

注意，在 settings.py 文件 INSTALLED_APPS 一项下增加 addressesapp，方法跟之前添加 nameapp 相同。下一节，我们讲怎么实现地址簿的主要功能。所有代码均已放到作者的 GitHub 仓库 chapter_6 文件夹（https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_6）。

6.2 编写应用——Django 最重要的功能

创建存储邮件地址的 Web 应用，我们不仅要建数据表存储数据，还要编写网页，支持用户新增、删除和查看地址簿。我们甚至还想将地址簿转换为电子表格或通过因特网将数据发送给其他应用。所有这些操作，Django 均提供了相应的功能（model、view、admin 和 API REST 框架）。我们先来讨论数据的存储方式。

6.2.1 model

创建电子邮件地址簿，我们需要将每位联系人的名字和邮件地址存储到数据表中。在 Django 中，数据表叫作 model，它是在 models.py 中定义的：

```
from django.db import models
from django.utils.translation import ugettext_lazy as _

class Person(models.Model):
    name = models.CharField(_('Name'), max_length=255, unique=True)
    mail = models.EmailField(max_length=255, blank=True)
    #display name on admin panel
    def __unicode__(self):
          return self.name
```

在 Django 中，数据表的列对应 model 的字段，model 的字段支持多种数据类型：整型、

字符型等。Django 自动为每个新添加的对象赋予一个自增的 ID 字段。上面代码，定义 name 字段时用到的 unique 选项，表示该 model 不允许联系人名称重复，blank 表示是否允许该字段为空，即用户是否可以不填该项。__unicode__函数可有可无，它的作用是将每个联系人对象表示为字符串形式（我们用联系人名称表示一个对象）。

创建 model 之后，我们需要在 SQLite 数据库将 model 表示的数据结构建立起来：

```
python manage.py makemigrations
python manage.py migrate
```

makemigrations 将模型的变动添加到迁移文件（addressesapp 文件夹下的 migrations 文件夹），migrate 将变动应用于数据库模式（database schema）。我们这个例子，同一个网站有多个应用，因此，实现迁移的命令应该加上应用的名称：python manage.py makemigrations 'appname'。

6.2.2 HTML 网页背后的 URL 和 view

我们已知道怎么存储数据，我们还需要用网页记录联系方式，并在另一个页面展示所有联系方式。下一节，我们简要介绍 HTML 页面的主要特性。

HTML 页面

本节讲解的所有代码都位于 test_server 文件夹下的模板文件夹（templates）中。

地址簿应用的主页，允许用户记录一条新的联系方式，主页见图 6.1：

图 6.1

如图 6.1 所示，网页的主体部分，有两个等待用户输入的输入框：联系人名称和邮件地址。单击 **Add** 按钮，这两项内容将被添加到数据库。该页面的 HTML 模板文件 home.html 如下所示：

```
{% extends "addressesapp/base.html" %}

{% block content %}
        <form action="" method="POST">
            {% csrf_token %}
            <h2 align = Center>Add person to address book </h2>
            <p> <br><br></p>
            <p align = Center><input type="search" class="span3"
              placeholder="person" name="name" id="search"
              autofocus /> </p>
            <p align = Center><input type="search" class="span3"
              placeholder="email" name="email" id="search"
              autofocus /> </p>
            <p align = Center><button type="submit" class="btn
              btn-primary btn-large pull-center">Add
              &raquo;</button></p>
        </form>
{% endblock content %}
```

我们用 POST 方法，将两个段落域（<p>……</p>）收集到的数据提交到服务器，提交事件是由 **Add** 按钮（&raquo: 在文本后面添加小箭头图标）激活的。网页的标题 **Add person to address book**（新增联系人到地址簿）用第二级标题渲染（<h2>……</h2>）。注意添加 csrf_token 标签，可启用跨站请求伪造攻击保护（更多内容请见 https://www.squarefree.com/securitytips/web-developers.html#CSRF）。

网页的样式（CSS 和 JavaScript 文件）以及页面底部、头部导航栏（**Home**、**Emails Book** 和 **Find** 按钮）是在 base.html 文件（见 templates 文件夹）中定义的。**Find** 功能用如下表单实现：

```
<form class="navbar-search pull-left" action="{% url 'get_contacts'
%}" method="GET">
          {% csrf_token %}
           <div style="overflow: hidden; padding-right: .5em;">
             <input type="text" name="term" style="width: 70%;" />
             <input type="submit" name="search" value="Find"
               size="30" style="float: right" />
           </div>
        </form>
```

表单中用 div 标签定义文本域，Find 按钮激活 GET 方法，请求一个 URL，服务器将 URL 交给 urls.py 文件里定义的 get_contacts 函数处理（见下节）。

另一个网页是展示地址簿的，如图 6.2 所示。

图 6.2

```
{% extends "addressesapp/base.html" %}

{% block content %}
<h2 align = Center>Email address book</h2>
<P align=Center>[
{% for letter in alphabet %}
which is given by the book.html file:
{% extends "addressesapp/base.html" %}

{% block content %}
<h2 align = Center>Email address book</h2>
<P align=Center>[
{% for letter in alphabet %}
<a href="{% url 'addressesbook' %}?letter={{letter}}" > {{letter}} </
a>
{% endfor %}
|<a href="addressesapp/book.html"> Index </a> ] </P>
<section id="gridSystem">
{% for contact in contacts %}①
<div class="row show-grid">
   <p align = Center><strong> name: </strong>{{ contact.name }}
<strong>email:</strong> {{ contact.mail }}    
        <a class="right" href="{% url 'delete_person' contact.name
%}" >   delete </a>
   </p>
</div>
{% endfor %}
</section>

{% endblock content %}
```

① 从前面“whichis qivew”到该行应删书，原书此处有误。可参考作者在 Github 上给出的代码。——译者注

再次提醒，上述代码用 base.html 渲染头部导航栏、页面底部和样式。标题 **Email address book** 用二级标题渲染，随后 for 循环{% for letter in alphabet %}，遍历 26 个英文字母，我们用它来实现只显示姓名以某个字母开头的联系人信息。具体实现方法是：将要查询的字母{{letter}}放到请求地址簿的 URL 之中进行查询。下面的代码，遍历联系人列表{% for contact in contacts %}，将列表渲染到页面：将每条联系人信息的姓名、邮件地址和按钮放到一个段落标记中，按钮的作用是从数据库删除联系人信息的。我们接下来讨论网页各事件的实现（添加、查找或删除联系人，展示地址簿）。

6.2.3　URL 声明和 view

我们现在讲解 urls.py 和 views.py 这两个文件是怎样与每个网页的 HTML 代码一道工作，实现我们想要的功能的。

由前面内容可知地址簿应用的两个主要网页 home 和 address book 各有一个 URL，在 Django 中这两个 URL 是在 urls.py 文件里面声明的：

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
from addressesapp.api import AddressesList

urlpatterns = patterns('',
    url(r'^docs/', include('rest_framework_swagger.urls')),
    url(r'^$','addressesapp.views.main'),
    url(r'^book/','addressesapp.views.addressesbook',name='addressesbo
ok'),
    url(r'^delete/(?P<name>.*)/','addressesapp.views.delete_person',
name='delete_person'),
    url(r'^book-search/','addressesapp.views.get_contacts', name='get_
contacts'),
    url(r'^addresses-list/', AddressesList.as_view(), name='addresseslist'),
    url(r'^notfound/','addressesapp.views.notfound',name='notfound'),
    url(r'^admin/', include(admin.site.urls)),
)
```

每个 URL 都是用正则表达式指定的（URL 字符串前要加一个 r），因此主页的地址被指定为 http://127.0.0.1:8000/（正则表达式以表示开始的^符号开始，后面紧跟表示结束的$符号），它的动作（新增一条联系人信息）由 views.py 文件里的 main 函数实现：

```
def main(request):
    context={}
```

```
    if request.method == 'POST':
        post_data = request.POST
        data = {}
        data['name'] = post_data.get('name', None)
        data['email'] = post_data.get('email', None)
        if data:
            return redirect('%s?%s' % (reverse('addressesapp.views.
main'),
                                      urllib.urlencode({'q': data})))
    elif request.method == 'GET':
        get_data = request.GET
        data= get_data.get('q',None)
        if not data:
            return render_to_response(
                'addressesapp/home.html', RequestContext(request,
context))
        data = literal_eval(get_data.get('q',None))
        print data
        if not data['name'] and not data['email']:
            return render_to_response(
                'addressesapp/home.html', RequestContext(request,
context))

        #add person to emails address book or update
        if Person.objects.filter(name=data['name']).exists():
            p = Person.objects.get(name=data['name'])
            p.mail=data['email']
            p.save()
        else:
            p = Person()
            p.name=data['name']
            p.mail=data['email']
            p.save()

        #restart page
        return render_to_response(
            'addressesapp/home.html', RequestContext(request,
               context))
```

每当用户提交、保存一条新的联系信息，POST 方法将请求转交给 GET 方法处理。如果姓名和邮件信息齐全，则创建一个 Person 对象，如果该对象已存在，则进行更新。注意名字相同，大小写不同，该方法（Person.objects.get()）将其看作是不同的名字，因此 Andrea、

ANDREA 和 andrea 将被当作三条不同的联系人信息。如想解决这个问题，读者可用 lower()函数将 name 字段的值转换为小写，这样三个 andrea 表达式指向同一个 andrea。

base.html 文件的查找动作对应 http://127.0.0.1:8000/book-search/这个 URL 地址，该动作的处理函数为 views.py 的 get_contacts 函数：

```
def get_contacts(request):
    logging.debug('here')
    if request.method == 'GET':
        get_data = request.GET
        data= get_data.get('term','')
        if data == '':
            return render_to_response(
               'addressesapp/nopersonfound.html',
                 RequestContext(request, {}))
        else:
            return redirect('%s?%s' %
        (reverse('addressesapp.views.addressesbook'),
    urllib.urlencode({'letter': data})))
```

如果用户在页面头部的文本域，输入一个非空字符串，该函数将请求转交 addressesbook 函数，去搜索用户输入的查询词（若未找到，展示相应的提示页）。

导航栏的 **Emails book** 按钮，指向 URL http://127.0.0.1:8000/book/，该 URL 触发 addressesbook 函数，展示联系人列表：

```
def addressesbook(request):
    context = {}
    logging.debug('address book')
    get_data = request.GET
    letter = get_data.get('letter',None)
    if letter:
        contacts = Person.objects.filter(name__iregex=r"(^|\s)%s" %
letter)
    else:
        contacts = Person.objects.all()
    #sorted alphabetically
    contacts = sort_lower(contacts,"name")#contacts.order_by("name")
    context['contacts']=contacts
    alphabetstring='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    context['alphabet']=[l for l in alphabetstring]
    return render_to_response(
        'addressesapp/book.html', RequestContext(request, context))
def sort_lower(lst, key_name):
```

```
    return sorted(lst, key=lambda item: getattr(item,
key_name).lower())
```

letter 字段存储名字（**Find** 按钮发起请求，再转交 addressesbook 函数处理时）或字母（从 Emails book 页面发起的请求），然后对 Person model 进行查找。检索得到的联系人信息存储在 context 对象 contacts 中，而字母存储在 context 对象 alphabet 中。如果没有指定字母，返回数据库所有联系人信息。请注意，名字首字母大小写都有可能，因此常用的 order_by 方法无法按照字母顺序为名字排序。因而，我们使用 sort_lower 方法，先将每个名字转换为小写形式，再按字母顺序排序。

删除操作由 delete_person 函数实现，请求 http://127.0.0.1:8000/delete/(?P.*)/这个 URL 地址可触发删除操作。.*表示所有字符都是名字的合法组成部分（注意如果我们只想用字符、数字和空格，应使用[a-zA-Z0-9]+）：

```
def delete_person(request,name):
    if Person.objects.filter(name=name).exists():
       p = Person.objects.get(name=name)
       p.delete()

    context = {}
    contacts = Person.objects.all()
    #sorted alphabetically
    contacts = sort_lower(contacts,"name")#contacts.order_by("name")
    context['contacts']=contacts
    return render_to_response(
        'addressesapp/book.html', RequestContext(request, context))
```

该函数搜索 Person 数据表，查询变量 name，找到后删除，最后返回只包含剩余联系人信息的地址簿页面。

同理，我们在 urls.py 文件中定义的“notfound”这一 URL，激活未找到联系人时的处理函数 notfound，你现在应该明白它的工作方式。

URL 地址 admin 指向 Django 的管理后台（见下节），而 docs 为 REST 框架的 swagger，我们在 6.3.3 节会讲。

6.3 管理后台

admin 面板是管理应用的用户界面，可通过浏览器访问。我们可以把刚创建的 model 加到 admin.py 文件里，命令如下：

```
from models import Person
admin.site.register(Person)
```

所有 model 都可以从用户界面访问，地址在：

```
http://127.0.0.1:8000/admin/
```

访问以上网址，需输入用户名和密码才能完成登录。所以，我们先用以下命令创建用户名，设置密码：

```
python manage.py createsuperuser
```

然后，输入用户名、密码（我用的是 andrea/a）。

登录后，我们就可以探索管理面板了，见图 6.3：

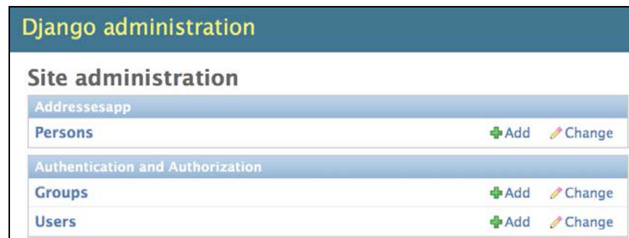


图 6.3

单击 **Persons**，我们将会看到用联系人名字表示的 Person 对象（因为我们在 Person model 的__unicode__函数里，指定用名字这个字段指代模型），见图 6.4：

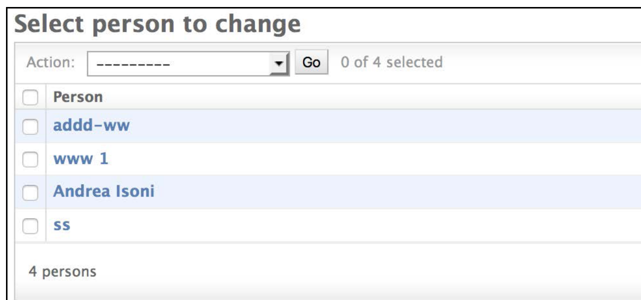


图 6.4

6.3.1　shell 接口

Django 框架还提供用 shell 探索和测试已创建的 model。在终端输入以下命令，启用 shell：

```
python manage.py shell
```

导入 Person model，并尝试操作它：

```
In [1]: from addressesapp.models import Person
In [2]: newcontact = Person()
In [3]: newcontact.name = 'myfriend1'
In [4]: newcontact.mail = 'bla@.com'
In [5]: newcontact.save()
In [6]: Person.objects.all()
Out[6]: [<Person: ss>, <Person: Andrea Isoni>, <Person: www 1>,
<Person: addd-ww>, <Person: myfriend1>]
```

上述代码新建一个联系人对象 myfriend1，然后查询所有的 Person 对象，以证实新对象有没有创建成功。

6.3.2 命令

Django 框架很灵活，我们可以自定义命令并用 manage.py 模块来执行。例如，我们想导出联系人列表到 CSV 文件。方法如下：在 management 文件夹创建 commands 文件夹（每个文件夹都放一个__init__.py 文件）。然后，扩展 BaseCommand 类，实现用自定义的命令导出联系人列表到 CSV 文件的功能：

```
from addressesapp.models import Person
from django.core.management.base import BaseCommand, CommandError
from optparse import make_option
import csv

class Command(BaseCommand):
    option_list = BaseCommand.option_list + (
                make_option('--output',
                             dest='output', type='string',
                             action='store',

                            help='output file'),
    )

def person_data(self, person):
        return [person.name,person.mail]

def handle(self, *args, **options):
     outputfile = options['output']
     contacts = Person.objects.all()
```

```
        header = ['Name','email']
        f = open(outputfile,'wb')
        writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC)
        writer.writerow(header)
        for person in contacts:
            writer.writerow(self.person_data(person))
```

该命令必须定义一个 handler 函数，执行导出操作。在 test_server 文件夹路径下，执行下面命令：

```
python manage.py contacts_tocsv -output='contacts_list.csv'
```

6.3.3　RESTful 应用编程接口（API）

RESTful API 是采用 HTTP 请求（比如 GET 和 POST）管理 Web 应用数据的应用编程接口。举个例子，我们要实现用 curl 命令调用 API 来获取地址簿。方法如下：我们需要在 settings.py INSTALLED_APPS 部分定义 rest_framework 应用，然后在 api.py 文件实现这个 API：

```
from rest_framework import viewsets, generics, views
from rest_framework.response import Response
from rest_framework.permissions import AllowAny
from rest_framework.pagination import PageNumberPagination
from addressesapp.serializers import AddressesSerializer
from addressesapp.models import Person

class LargeResultsSetPagination(PageNumberPagination):
    page_size = 1000
    page_size_query_param = 'page_size'
    max_page_size = 10000

class AddressesList(generics.ListAPIView):

    serializer_class = AddressesSerializer
    permission_classes = (AllowAny,)
    pagination_class = LargeResultsSetPagination

    def get_queryset(self):
        query = self.request.query_params.get
        if query('name'):
           return Person.objects.filter(name=query('name'))
        else:
           return Person.objects.all()
```

我们用 ListAPIView 类，返回所有 Person 对象，或只返回与 name 值匹配的 Person 对象。因

为返回的列表也许很大，我们需要重写 PageNumberPagination 类，在同一页展示更多的对象；LargeResultsSetPagination 类，每页最多可包含 1 万个对象。该 API 需要将 Person 对象转换为 JSON 格式的对象，在 serializers.py 文件编写 serializer 对象 Addresses Serializer 来实现这个功能：

```
from addressesapp.models import Person
from rest_framework import serializers

class AddressesSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:
        model = Person
        fields = ('id', 'name', 'mail')
```

然后，我们可以用 curl 命令请求地址簿：

```
curl -X GET http://localhost:8000/addresses-list/
```

注意 URL 最后的斜线。同样，我们可以通过指定联系人姓名来获取他们的邮箱：

```
curl -X GET http://localhost:8000/addresses-st/?name=name_value
```

如果联系人数量很多（或修改分页的数量），我们可以用参数指定请求哪一页。在 urls.py 文件，我们还指定了 Swagger RESTful API 文档的 URL 地址，开发人员可在浏览器查看和测试这些 API，见图 6.5：

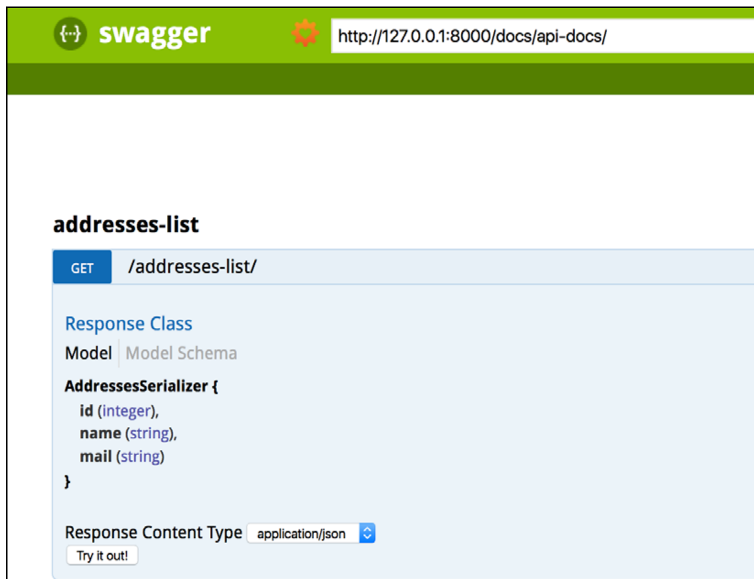


图 6.5

这种方式非常友好，便于开发人员验证 API 是否能够正常工作，数据格式是否正确。

6.4 小结

本章讨论怎样用 Django 框架搭建 Web 应用。我们介绍了 model、admin、view、command、shell 以及 RESTful API 等 Django 的主要功能。学完本章，读者应该已具备开发实用型 Web 应用的必要知识。

后续两章，我们将用这些知识和前面所讲的内容搭建电影推荐引擎和电影情感分析系统。

第 7 章
电影推荐系统 Web 应用

本章讲解如何用 Django 框架搭建一个真实的推荐系统。该系统的主要功能是，根据用户的喜好（第 5 章讲过），向订阅了推荐服务的用户推荐电影，我们沿用第 5 章的电影评分数据：603 部电影的打分数据，其中每部电影的打分人次在 50 次以上，共有 942 位用户参与打分。为了能够得到系统提供的电影推荐服务，每位用户需要为一定数量的电影打分，我们特意讲解信息检索系统的实现（第 4 章），以帮助用户检索电影、完成打分。我们还将讨论 Django 应用的不同部分：setting、model、用户登录/登出、命令、信息检索系统、推荐系统、管理后台和 API（所有代码均已放到作者 GitHub 仓库 chapter_7 文件夹：https://github.com/ai2010/machine_learning_for_the_web/tree/master/chapter_7）。因为第 6 章只介绍了 Django 的主要功能，所以在开发电影推荐系统过程中，每遇到一个新功能，我们都会讲解其技术细节。现在我们开始介绍推荐系统这个 Web 应用的初始化配置，让它先跑起来。

7.1 让应用跑起来

按照之前的做法，新建一个 Django 项目：

```
django-admin startproject server_movierecsys
```

在 server_movierecsys 文件夹下，新建一个应用：

```
python manage.py startapp books_recsys_app
```

接下来，对 settings.py 里面的各项内容进行设置。仿照第 6 章的做法，安装应用，指定 HTML 模板和样式文件的路径，指定使用 SQLite 数据库：

```
INSTALLED_APPS = (
```

```
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework_swagger',
    'books_recsys_app',
)

TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)
STATIC_URL = '/static/'
STATICFILES_DIRS = ( os.path.join(BASE_DIR, "static"), )
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

除了 Django 提供的标准应用，我们还在应用列表（INSTALLED_APPS）添加了 REST 框架（swagger）。

该例中，我们需要将数据加载到内存中长久保存，这样每次用户请求数据，无须计算或检索数据，以此来提升用户体验。我们在 settings.py 文件，设置启用 Django 框架的缓存系统，将数据或计算开销较大的运算结果保存到内存：

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.
FileBasedCache',
        'LOCATION': '/var/tmp/django_cache',
        'TIMEOUT': None,
    }
}
```

我们选择使用存储在/var/tmp/django_cache 的 **File Based Cache** 类型，TIMEOUT（过期时间）设置为 None，表示缓存中的数据永不过期。

用以下命令创建超级管理员用户，以便使用管理后台：

```
python manage.py createsuperuser (admin/admin)
```

输入以下命令，启动服务器，访问 http://localhost:8000/，就能看到应用的首页：

```
python manage.py runserver
```

7.2 model

对于电影推荐系统这一应用，我们需要存储每一部电影的相关数据以及每位网站用户的打分数据。我们需要创建三个模型：

```
class UserProfile(models.Model):
    user = models.ForeignKey(User, unique=True)
    array = jsonfield.JSONField()
    arrayratedmoviesindxs = jsonfield.JSONField()
    lastrecs = jsonfield.JSONField()

    def __unicode__(self):
            return self.user.username

    def save(self, *args, **kwargs):
        create = kwargs.pop('create', None)
        recsvec = kwargs.pop('recsvec', None)
        print 'create:',create
        if create==True:
            super(UserProfile, self).save(*args, **kwargs)
        elif recsvec!=None:
             self.lastrecs = json.dumps(recsvec.tolist())
             super(UserProfile, self).save(*args, **kwargs)
        else:
            nmovies = MovieData.objects.count()
            array = np.zeros(nmovies)
            ratedmovies = self.ratedmovies.all()
            self.arrayratedmoviesindxs = json.dumps([m.movieindx for m
in ratedmovies])
            for m in ratedmovies:
                array[m.movieindx] = m.value
            self.array = json.dumps(array.tolist())
            super(UserProfile, self).save(*args, **kwargs)
```

```
class MovieRated(models.Model):
    user = models.ForeignKey(UserProfile, related_name='ratedmovies')
    movie = models.CharField(max_length=100)
    movieindx = models.IntegerField(default=-1)
    value = models.IntegerField()

class MovieData(models.Model):
    title = models.CharField(max_length=100)
    array = jsonfield.JSONField()
    ndim = models.IntegerField(default=300)
    description = models.TextField()
```

MovieData 这个 model 存储每部电影的相关数据：影片名称、简介、向量表示（ndim 为向量的维度）。MovieRated 记录每位用户为哪些电影打过分（每个 MovieRated 对象，对应一个 UserProfile 对象，即使用该网站的用户）。UserProfile model 存储网站所有注册用户，有这个对象，电影打分和推荐功能才有存在的可能。每个 UserProfile 是对 Django 自带的 user model 的扩展，在此基础上增加了 array 字段，以存储当前用户对所有电影的打分数据，此外还增加了 recsvec 字段，存储上一次为当前用户推荐的电影：我们重写了 save 函数，用（与当前用户对应的）MovieRated 对象填充 array 字段（如果 else 语句为真）；用上一次的推荐数据填充 lastrecs（如果 else if 语句为真）。请注意，MovieRated model 的外键 UserProfile，其 related_name 为“raetedmovies”：UserProfile model 的 save 函数中，self.ratedmovies.all()指的是同一 UserProfile 的所有 RatedMovie 对象。UserProfile model 的 arrayratedmoviesindxs 字段，记录着用户打过分的所有电影，电影推荐系统应用的 API 将使用这些数据。

我们执行以下命令，将 models.py 文件中定义的这些数据结构写到数据库：

```
python manage.py makemigrations
python manage.py migrate
```

7.3 命令

电影推荐系统这一应用，为了让用户感觉速度很快，我们想将数据加载到内存（缓存），这就要用到自定义的命令。我们这里用的电影数据（603 部电影，942 名用户的打分数据，每部电影打分人次在 50 次以上）跟第 4 章使用的相同，但是除此之外，搭建信息检索系统，让用户检索电影、完成打分，还需要向用户提供电影简介供其参考。我们接下来开发第一条命令，实现从第 4 章的效用矩阵获取电影名称，再从开放电影数据库（Open Movie

Database，OMDB）采集电影简介的功能：

```
from django.core.management.base import BaseCommand
import os
import optparse
import numpy as np
import json
import pandas as pd
import requests
class Command(BaseCommand):

    option_list = BaseCommand.option_list + (
            optparse.make_option('-i', '--input', dest='umatrixfile',
                                 type='string', action='store',
                                 help=('Input utility matrix')),
            optparse.make_option('-o', '--outputplots',
dest='plotsfile',
                                 type='string', action='store',
                                 help=('output file')),
            optparse.make_option('--om', '--outputumatrix',
dest='umatrixoutfile',
                                 type='string', action='store',
                                 help=('output file')),
        )

    def getplotfromomdb(self,col,df_moviesplots,df_movies,df_
utilitymatrix):
        string = col.split(';')[0]

        title=string[:-6].strip()
        year = string[-5:-1]
        plot = ' '.join(title.split(' ')).encode('ascii','ignore')+'.'

        url = "http://www.omdbapi.com/?t="+title+"&y="+year+"&plot=fu
ll&r=json"

        headers={"User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2049.0
Safari/537.36"}
        r = requests.get(url,headers=headers)
        jsondata = json.loads(r.content)
        if 'Plot' in jsondata:
            #store plot + title
```

```
                plot += jsondata['Plot'].encode('ascii','ignore')

            if plot!=None and plot!='' and plot!=np.nan and
    len(plot)>3:#at least 3 letters to consider the movie
                df_moviesplots.loc[len(df_moviesplots)]=[string,plot]
                df_utilitymatrix[col] = df_movies[col]
                print len(df_utilitymatrix.columns)

            return df_moviesplots,df_utilitymatrix

        def handle(self, *args, **options):
            pathutilitymatrix = options['umatrixfile']
            df_movies = pd.read_csv(pathutilitymatrix)
            movieslist = list(df_movies.columns[1:])

            df_moviesplots = pd.DataFrame(columns=['title','plot'])
            df_utilitymatrix = pd.DataFrame()
            df_utilitymatrix['user'] = df_movies['user']

            for m in movieslist[:]:
                df_moviesplots,df_utilitymatrix=self.getplotfromomdb(m,df_
    moviesplots,df_movies,df_utilitymatrix)

            outputfile = options['plotsfile']
            df_moviesplots.to_csv(outputfile, index=False)
            outumatrixfile = options['umatrixoutfile']
            df_utilitymatrix.to_csv(outumatrixfile, index=False)
```

该命令的执行方式如下：

```
python manage.py --input=utilitymatrix.csv --outputplots=plots.csv –
outputumatrix='umatrix.csv'
```

每部电影的名称存储在 utilitymatrix 文件，getplotfromomdb 函数用 Python 的 requests 模块，从网站 http://www.omdbapi.com/检索电影的简介。电影简介（和名称）以及相应的效用矩阵（outputumatrix）保存到一个 CSV 文件（outputplots）。

我们再来看第 2 个命令：用电影简介，创建信息检索系统（TF-IDF 模型），支持根据用户输入的词语查找相关电影。然后，将 TF-IDF 模型和最初的推荐系统模型（基于商品的 CF 和对数似然比）保存到 Django 的缓存之中。代码如下：

```
from django.core.management.base import BaseCommand
import os
```

```
import optparse
import numpy as np
import pandas as pd
import math
import json
import copy
from BeautifulSoup import BeautifulSoup
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import WordPunctTokenizer
tknzr = WordPunctTokenizer()
#nltk.download('stopwords')
stoplist = stopwords.words('english')
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
from sklearn.feature_extraction.text import TfidfVectorizer
from books_recsys_app.models import MovieData
from django.core.cache import cache

class Command(BaseCommand):

    option_list = BaseCommand.option_list + (
            optparse.make_option('-i', '--input', dest='input',
                                 type='string', action='store',
                                 help=('Input plots file')),
            optparse.make_option('--nmaxwords', '--nmaxwords',
dest='nmaxwords',
                                 type='int', action='store',
                                 help=('nmaxwords')),
            optparse.make_option('--umatrixfile', '--umatrixfile',
dest='umatrixfile',
                                 type='string', action='store',
                                 help=('umatrixfile')),
    )

def PreprocessTfidf(self,texts,stoplist=[],stem=False):
    newtexts = []
    for i in xrange(len(texts)):
        text = texts[i]
        if stem:
           tmp = [w for w in tknzr.tokenize(text) if w not in
stoplist]
        else:
```

```
                tmp = [stemmer.stem(w) for w in [w for w in tknzr.
tokenize(text) if w not in stoplist]]
            newtexts.append(' '.join(tmp))
        return newtexts

    def handle(self, *args, **options):
        input_file = options['input']

        df = pd.read_csv(input_file)
        tot_textplots = df['plot'].tolist()
        tot_titles = df['title'].tolist()
        nmaxwords=options['nmaxwords']
        vectorizer = TfidfVectorizer(min_df=0,max_features=nmaxwords)
        processed_plots = self.PreprocessTfidf(tot_
textplots,stoplist,True)
        mod_tfidf = vectorizer.fit(processed_plots)
        vec_tfidf = mod_tfidf.transform(processed_plots)
        ndims = len(mod_tfidf.get_feature_names())
        nmovies = len(tot_titles[:])

        #delete all data
        MovieData.objects.all().delete()

        matr = np.empty([1,ndims])
        titles = []
        cnt=0
        for m in xrange(nmovies):
            moviedata = MovieData()
            moviedata.title=tot_titles[m]
            moviedata.description=tot_textplots[m]
            moviedata.ndim= ndims
            moviedata.array=json.dumps(vec_tfidf[m].toarray()[0].
tolist())
            moviedata.save()
            newrow = moviedata.array
            if cnt==0:
                matr[0]=newrow
            else:
                matr = np.vstack([matr, newrow])
            titles.append(moviedata.title)
            cnt+=1
        #cached
        cache.set('data', matr)
        cache.set('titles', titles)
```

```
        cache.set('model',mod_tfidf)

        #load the utility matrix
        umatrixfile = options['umatrixfile']
        df_umatrix = pd.read_csv(umatrixfile)
        Umatrix = df_umatrix.values[:,1:]
        cache.set('umatrix',Umatrix)
        #load rec methods...
        cf_itembased = CF_itembased(Umatrix)
        cache.set('cf_itembased',cf_itembased)
        llr = LogLikelihood(Umatrix,titles)
        cache.set('loglikelihood',llr)

from scipy.stats import pearsonr
from scipy.spatial.distance import cosine
def sim(x,y,metric='cos'):
    if metric == 'cos':
       return 1.-cosine(x,y)
    else:#correlation
       return pearsonr(x,y)[0]

class CF_itembased(object):
...
class LogLikelihood(object):
...
```

命令的执行方式如下：

```
python manage.py load_data --input=plots.csv --nmaxwords=30000
--umatrixfile=umatrix.csv
```

参数 input 接收的是用 get_plotsfromtitles 命令获取到的电影简介。用参数 nmaxwords 指定最大单词数，创建 TF-IDF 模型（见第 4 章）。将每部电影的数据保存到 MovieData 对象（电影名称、TF-IDF 表示、简介、TF-IDF 词汇量）。请注意，第一次运行该命令，需要用 nltk.download('stopwords')语句下载停用词表（上述代码，该行语句被注释掉了）。

用以下命令，将 TF-IDF 模型、电影名称列表和用 TF-IDF 矩阵表示的电影，保存到 Django 的缓存：

```
from django.core.cache import cache
...
cache.set('model',mod_tfidf)
```

```
cache.set('data', matr)
cache.set('titles', titles)
```



请注意，Django 的 cache 模块（django.core.cache）使用前，要先导入它。导入语句要放到文件的开始位置。

照旧，我们用效用矩阵（umatrixfile）初始化两种推荐方法：基于商品的协同过滤和对数似然比方法。这两种方法没有写到上面的代码中，因为它们基本与第 5 章讲解的相同（完整代码照例见作者的 GitHub 仓库 chapter_7 文件夹）。然后，我们将这两种推荐模型及效用矩阵加载到 Django 的缓存以备后续之用：

```
cache.set('umatrix',Umatrix)
   cache.set('cf_itembased',cf_itembased)
   cache.set('loglikelihood',llr)
```

现在，调用相应的名称，就可以在网页使用数据（和模型），详见后续几节。

7.4 实现用户的注册、登录和登出功能

电影推荐系统这一应用，是要向注册网站的不同用户推荐电影，因此我们需要实现注册、登录和登出功能。我们使用 Django 的标准 User 对象实现注册过程，前面 *model* 一节已讲过 User 对象。该应用的每个网页都将引用 base.html 页面，我们在 base.html 实现了顶部横栏，支持用户注册或登录（右侧），见图 7.1：



图 7.1

单击 **sign in**（登录）或 **sign up**（注册）按钮，将激活以下代码：

```
                <form class="navbar-search pull-right" action="{% url
'auth' %}" method="GET">
                  {% csrf_token %}
                   <div style="overflow: hidden; padding-right:
.5em;">
                     <input type="submit" name="auth_method"
value="sign up" size="30" style="float: right" />
                     <input type="submit" name="auth_method"
value="sign in" size="30" style="float: right" />
```

```
            </div>
        </form>
```

这两个方法指向 urls.py：

```
url(r'^auth/', 'books_recsys_app.views.auth', name='auth')
```

上述 URL 调用 views.py 文件的 auth 函数来处理：

```
def auth(request):
    if request.method == 'GET':
        data = request.GET
        auth_method = data.get('auth_method')
        if auth_method=='sign in':
           return render_to_response(
               'books_recsys_app/signin.html', RequestContext(request,
{}))
        else:
            return render_to_response(
                'books_recsys_app/createuser.html',
RequestContext(request, {}))
    elif request.method == 'POST':
        post_data = request.POST
        name = post_data.get('name', None)
        pwd = post_data.get('pwd', None)
        pwd1 = post_data.get('pwd1', None)
        create = post_data.get('create', None)#hidden input
        if name and pwd and create:
           if User.objects.filter(username=name).exists() or
pwd!=pwd1:
               return render_to_response(
                   'books_recsys_app/userexistsorproblem.html',
RequestContext(request))
           user = User.objects.create_user(username=name,password=pwd)
           uprofile = UserProfile()
           uprofile.user = user
           uprofile.name = user.username
           uprofile.save(create=True)

           user = authenticate(username=name, password=pwd)
           login(request, user)
           return render_to_response(
               'books_recsys_app/home.html', RequestContext(request))
```

```
        elif name and pwd:
            user = authenticate(username=name, password=pwd)
            if user:
                login(request, user)
                return render_to_response(
                    'books_recsys_app/home.html',
RequestContext(request))
            else:
                #notfound
                return render_to_response(
                    'books_recsys_app/nopersonfound.html',
                       RequestContext(request))
```

auth 函数将用户导向图 7.2 注册页面：

图 7.2

如用户已注册，将导向登录页面，见图 7.3：

图 7.3

该页面支持用户输入用户名和密码，登录网站。然后，网站用这些数据创建 Django 框架的 User 对象和相应的 UserProfile 对象（注意参数 create 为 True，表示保存新建的用户对

象时，没有为其指定电影列表，因为新用户还没有为任何电影打过分）：

```
user = User.objects.create_user(username=name,password=pwd)
uprofile = UserProfile()
uprofile.user = user
uprofile.save(create=True)
user = authenticate(username=name, password=pwd)
```

然后，用 Django 的标准登录函数将用户登录进来：

```
from django.contrib.auth import authenticate, login
...
login(request, user)
```

登录之后，网站的顶栏显示效果（username 为 **a**）见图 7.4：



图 7.4

如果某一用户的名字已存在（注册时，遇到的异常情况）或用户找不到（登录时，遇到的异常情况），这两种情况的处理均已实现，读者可自行查看代码，了解这些异常事件的处理方法。

sign out（登出）按钮指向 urls.py 以下语句：

```
url(r'^signout/','books_recsys_app.views.signout',name='signout')
```

该 URL 调用 views.py 文件的 signout 函数处理：

```
from django.contrib.auth import logout
…
def signout(request):
    logout(request)
    return render_to_response(
        'books_recsys_app/home.html', RequestContext(request))
```

该函数使用了 Django 标准的 logout 方法，将用户导向首页（再次显示 **sign in** 和 **sign up**①按钮）。实现了注册、登录和登出功能之后，用户可以登录网站，使用信息检索系统（搜索引擎）搜索电影，为电影打分。下节，我们讲信息检索系统的实现方法。

7.5 信息检索系统（电影查询）

用户使用图 7.5 的页面，搜索电影，以便为电影打分：

① 原文为“sign in and sign out”，用户登出后，应显示注册按钮“sign up”。——译者注

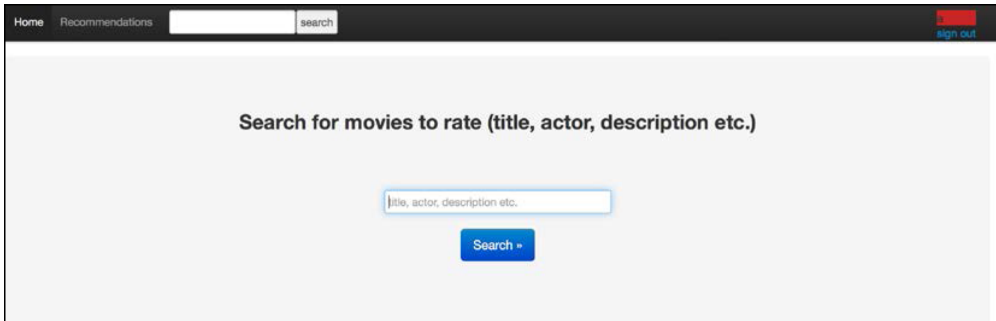


图 7.5

在文本框输入几个相关词，该页面（通过 urls.py 文件里的 URL home）调用 views.py 文件里的 home 函数：

```
def home(request):
    context={}
    if request.method == 'POST':
        post_data = request.POST
        data = {}
        data = post_data.get('data', None)
        if data:
            return redirect('%s?%s' % (reverse('books_recsys_app.
views.home'),
                                urllib.urlencode({'q': data})))
    elif request.method == 'GET':
        get_data = request.GET
        data = get_data.get('q',None)
        titles = cache.get('titles')
        if titles==None:
            print 'load data...'
            texts = []
            mobjs = MovieData.objects.all()
            ndim = mobjs[0].ndim
            matr = np.empty([1,ndim])
            titles_list = []
            cnt=0
            for obj in mobjs[:]:
                texts.append(obj.description)
                newrow = np.array(obj.array)
                #print 'enw:',newrow
                if cnt==0:
                    matr[0]=newrow
                else:
```

```
                matr = np.vstack([matr, newrow])
            titles_list.append(obj.title)
            cnt+=1
        vectorizer = TfidfVectorizer(min_df=1,max_features=ndim)
        processedtexts = PreprocessTfidf(texts,stoplist,True)
        model = vectorizer.fit(processedtexts)
        cache.set('model',model)
        #cache.set('processedtexts',processedtexts)
        cache.set('data', matr)
        cache.set('titles', titles_list)
    else:
        print 'loaded',str(len(titles))

    Umatrix = cache.get('umatrix')
    if Umatrix==None:
        df_umatrix = pd.read_csv(umatrixpath)
        Umatrix = df_umatrix.values[:,1:]
        cache.set('umatrix',Umatrix)
        cf_itembased = CF_itembased(Umatrix)
        cache.set('cf_itembased',cf_itembased)
        cache.set('loglikelihood',LogLikelihood(Umatrix,moviesli
st))
    if not data:
        return render_to_response(
            'books_recsys_app/home.html', RequestContext(request,
context))

    #load all movies vectors/titles
    matr = cache.get('data')
    titles = cache.get('titles')
    model_tfidf = cache.get('model')
    #find movies similar to the query
    queryvec = model_tfidf.transform([data.lower().
encode('ascii','ignore')]).toarray()
    sims= cosine_similarity(queryvec,matr)[0]
    indxs_sims = list(sims.argsort()[::-1])
    titles_query = list(np.array(titles)[indxs_sims]
[:nmoviesperquery])

    context['movies']= zip(titles_query,indxs_
sims[:nmoviesperquery])
    context['rates']=[1,2,3,4,5]
    return render_to_response(
```

```
                'books_recsys_app/query_results.html',
                 RequestContext(request, context))
```

函数开始位置的参数 data，用来存储用户输入的查询词。load_data 命令将模型加载到内存后，该函数利用这个模型，将用户的查询词转换为用 TF-IDF 方法表示的向量：

```
matr = cache.get('data')
titles = cache.get('titles')
model_tfidf = cache.get('model')
```

从缓存检索矩阵（键：matr）、电影名称（键：titles），返回与查询词向量相似的电影列表（更多内容见第 4 章）。第 1 次调用该函数时，缓存为空，这时直接创建模型（和其他数据），并将其加载到内存。至于检索系统怎么用，我们举个例子，比如用户输入 war 作为查询词，网站将返回与 war（战争）最相似的电影（query_results.html），见图 7.6：



图 7.6

图 7.6 为查询词 war 的检索结果，系统返回 5 部电影（我们可以在 views.py 文件开始位置用 nmoviesperquery 指定每次查询返回的电影数量），并且它们大多与战争有关。我们下节要实现在这个页面上为电影打分这一功能。

7.6　打分系统

在电影检索结果页面（见图 7.6），每位用户（登录后）单击电影名称后面的分数，即可为电影打分。单击分数的动作，将触发 views.py 文件的 rate_movie 函数（通过在 urls.py 定义的 URL）：

```
def rate_movie(request):
    data = request.GET
    rate = data.get("vote")
```

```
    movies,moviesindxs = zip(*literal_eval(data.get("movies")))
    movie = data.get("movie")
    movieindx = int(data.get("movieindx"))
    #save movie rate
    userprofile = None
    if request.user.is_superuser:
        return render_to_response(
            'books_recsys_app/superusersignin.html',
RequestContext(request))
    elif request.user.is_authenticated() :
        userprofile = UserProfile.objects.get(user=request.user)
    else:
        return render_to_response(
            'books_recsys_app/pleasesignin.html',
RequestContext(request))

    if MovieRated.objects.filter(movie=movie).
filter(user=userprofile).exists():
        mr = MovieRated.objects.get(movie=movie,user=userprofile)
        mr.value = int(rate)
        mr.save()
    else:
        mr = MovieRated()
        mr.user = userprofile
        mr.value = int(rate)
        mr.movie = movie
        mr.movieindx = movieindx
        mr.save()

    userprofile.save()
    #get back the remaining movies
    movies = RemoveFromList(movies,movie)
    moviesindxs = RemoveFromList(moviesindxs,movieindx)
    print movies
    context = {}
    context["movies"] = zip(movies,moviesindxs)
    context["rates"] = [1,2,3,4,5]
    return render_to_response(
        'books_recsys_app/query_results.html',
         RequestContext(request, context))
```

该函数将电影的分数存储在 MovieRated 对象中，同时更新用户相应的电影分数向量 rate（通过 userprofile.save()实现）。然后将没有打过分的电影发送给页面 query_results.html。

请注意，用户登录后才能为电影打分，若用户没有登录，则执行异常事件处理流程，要求用户登录（页面：pleasesignin.html）。

7.7 推荐系统

该函数将使用我们在 views.py 文件的前面定义的几个参数。

```
nminimumrates=5
numrecs=5
recmethod = 'loglikelihood'
```

nminimumrates 指定用户最少为几部电影打过分，才能得到推荐服务；numrecs 指定向用户推荐多少部电影；recmethod 指定推荐系统使用的推荐方法。用户单击顶栏的 **Recommendations**（电影推荐）可查看系统向他推荐的电影，见图 7.7：



图 7.7

该动作将触发 views.py 文件的 movies_recs 函数（通过请求在 urls.py 文件里定义相应的 URL）：

```
def movies_recs(request):

    userprofile = None
    if request.user.is_superuser:
        return render_to_response(
            'books_recsys_app/superusersignin.html',
RequestContext(request))
    elif request.user.is_authenticated():
        userprofile = UserProfile.objects.get(user=request.user)
    else:
        return render_to_response(
            'books_recsys_app/pleasesignin.html',
RequestContext(request))
        ratedmovies=userprofile.ratedmovies.all()
        context = {}
        if len(ratedmovies)<nminimumrates:
            context['nrates'] = len(ratedmovies)
            context['nminimumrates']=nminimumrates
            return render_to_response(
```

```
                    'books_recsys_app/underminimum.html',
RequestContext(request, context))

    u_vec = np.array(userprofile.array)
    Umatrix = cache.get('umatrix')
    movieslist = cache.get('titles')
    #recommendation...
    u_rec = None
    if recmethod == 'cf_userbased':
        u_rec = CF_userbased(u_vec,numrecs,Umatrix)
    elif recmethod == 'cf_itembased':
        cf_itembased = cache.get('cf_itembased')
        if cf_itembased == None:
            cf_itembased = CF_itembased(Umatrix)
        u_rec = cf_itembased.CalcRatings(u_vec,numrecs)
    elif recmethod == 'loglikelihood':
        llr = cache.get('loglikelihood')
        if llr == None:
            llr = LogLikelihood(Umatrix,movieslist)
        u_rec = llr.GetRecItems(u_vec,True)
    #save last recs
    userprofile.save(recsvec=u_rec)
    context['recs'] = list(np.array(movieslist)[list(u_rec)]
[:numrecs])
    return render_to_response(
        'books_recsys_app/recommendations.html',
          RequestContext(request, context))
```

该函数从当前用户的 UserProfile 对象检索所有已打分电影的向量，从缓存加载推荐系统方法（用 recmethod 参数指定）计算推荐哪些电影。推荐的电影首先存储到 userprofile 对象，然后返回给 recommendations.html 页面。例如，用 cf_itembased 方法得到的推荐列表，见图 7.8：

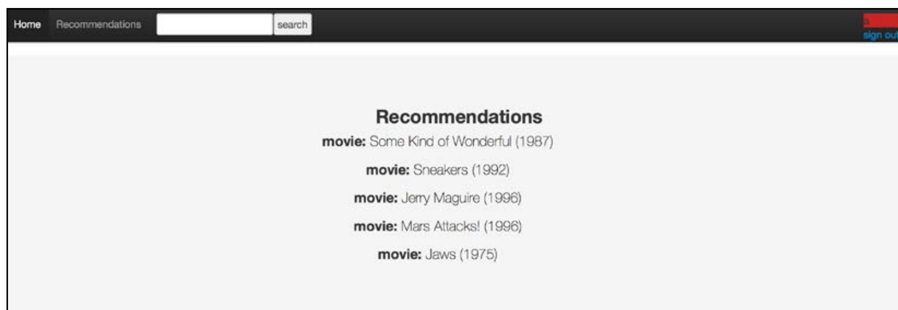


图 7.8

图 7.8 是推荐结果页面的示例，我们为与 war 关键词相关的 5 部电影打过分后（见之前的打分页面），得到了这样的推荐结果。读者可尝试修改前面设置的三个参数，用不同的推荐算法，看看结果有什么不同。

7.8 管理界面和 API

为了方便管理推荐系统的数据，我们可以使用管理后台和 API。编写 admin.py 文件，我们就能在管理面板看到电影数据和用户注册数据，代码如下：

```
from django.contrib import admin
from books_recsys_app.models import MovieData,UserProfile

class MoviesAdmin(admin.ModelAdmin):
      list_display = ['title', 'description']

admin.site.register(UserProfile)
admin.site.register(MovieData,MoviesAdmin)
```

然后，在 urls.py 文件指定 admin 的 URL：

```
url(r'^admin/', include(admin.site.urls))
```

现在，我们应该能够在管理面板（地址为 http://localhost:8000/admin/）看到两个模型及在 admin.py 文件里指定的、要在管理面板显示的各个 model 的字段，见图 7.9：

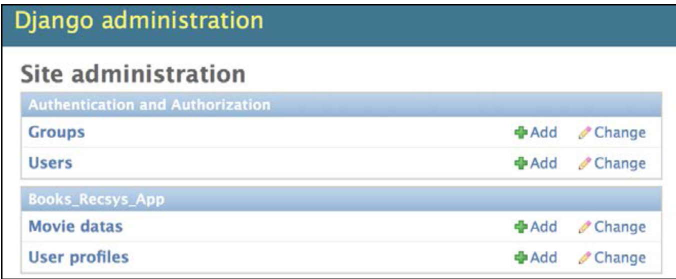


图 7.9

我们若要想实现通过 API 终点（endpoint）检索每位注册用户数据的功能，首先需要编写 serializers.py，指定检索 UserProfile model 的字段：

```
from books_recsys_app.models import UserProfile
```

```
from rest_framework import serializers

class UsersSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = UserProfile
        fields = ('name', 'arrayratedmoviesindxs','lastrecs')
```

比如，我们想采集用户打过分的电影的 ID、上一次向他推荐的电影 ID。然后，我们在 api.py 文件编写 API，如下所示：

```
from rest_framework import generics
from rest_framework.permissions import AllowAny
from rest_framework.pagination import PageNumberPagination
from books_recsys_app.serializers import UsersSerializer
from books_recsys_app.models import UserProfile

class LargeResultsSetPagination(PageNumberPagination):
    page_size = 1000
    page_size_query_param = 'page_size'
    max_page_size = 10000

class UsersList(generics.ListAPIView):

    serializer_class = UsersSerializer
    permission_classes = (AllowAny,)
    pagination_class = LargeResultsSetPagination

    def get_queryset(self):
        query = self.request.query_params.get
        if query('name'):
           return UserProfile.objects.filter(name=query('name'))
        else:
           return UserProfile.objects.all()
```

我们也许只想查询特定用户的数据，因此该 API 需要支持用 name 作为查询参数。编写好 API 后，我们再在 urls.py 文件设置相应的 URL：

```
url(r'^users-list/',UsersList.as_view(),name='users-list')
```

然后，我们可以在终端用 curl 命令调用这个 API 接口：

```
curl -X GET localhost:8000/users-list/
```

我们还可以用 swagger 接口调用这个 API 接口，进行测试（见第 6 章）。

7.9 小结

本章，我们介绍了如何用 Django 框架搭建电影推荐系统。学完本章，对于如何用 Python 语言编写用机器学习算法驱动的、专业的 Web 推荐应用，你应该具有一定的信心了。

下一章，也就是本书的最后一章，我们将通过另外一个实例——搭建 Web 情感分析系统，加深你对如何高效地运用 Python 语言编写 Web 机器学习应用的理解。

第 8 章
影评情感分析应用

在本章中，我们用前几章介绍的算法和方法，开发一套能够判断影评情感倾向的情感分析系统。我们还将用 **Scrapy** 库，通过搜索引擎 API（Bing 搜索引擎）从不同的网站采集影评数据。我们用 newspaper 库或预先定义好的 HTML 页面抽取规则，从影评数据抽取影评内容和电影名称。我们用朴素贝叶斯分类器，以包含分类信息最多（使用 X^2 检测）的词语作为特征，得到每条影评的情感倾向，第 4 章讲过该方法。我们用第 4 章讲过的 PageRank 算法，计算与每个电影查询词相关的网页次序。本章将讨论影评情感分析应用的代码，包括 Django 的 model 和 view，我们用 Scrapy 库的 scraper 从网页采集影评数据。我们首先给出 Web 应用的样例，解释我们使用的搜索引擎 API 和将其整合到应用的方法。然后，讲解影评的采集方法：将 Scrapy 库整合到 Django、编写存储数据的 model 和管理应用的主要命令。本章讨论的这些代码均已放到作者的 GitHub 仓库 chapter_8 文件夹，地址为 https://github.com/ ai2010/machine_learning_for_the_web/tree/master/chapter_8。

8.1　影评情感分析应用用法简介

首页展示效果见图 8.1：

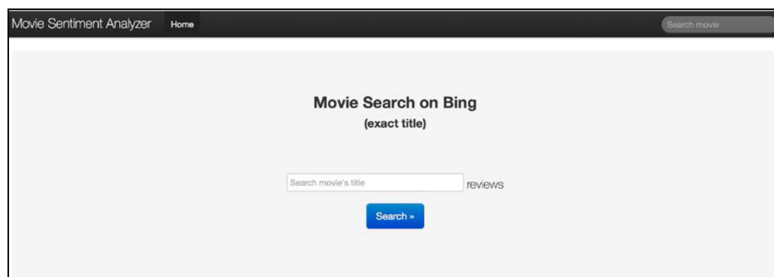


图 8.1

如果用户想知道影评的情感倾向和相关性，他们输入电影的名称进行查询即可。例如，图 8.2 显示的是电影 *Batman vs Superman Dawn of Justice*[①]影评情感分析结果：

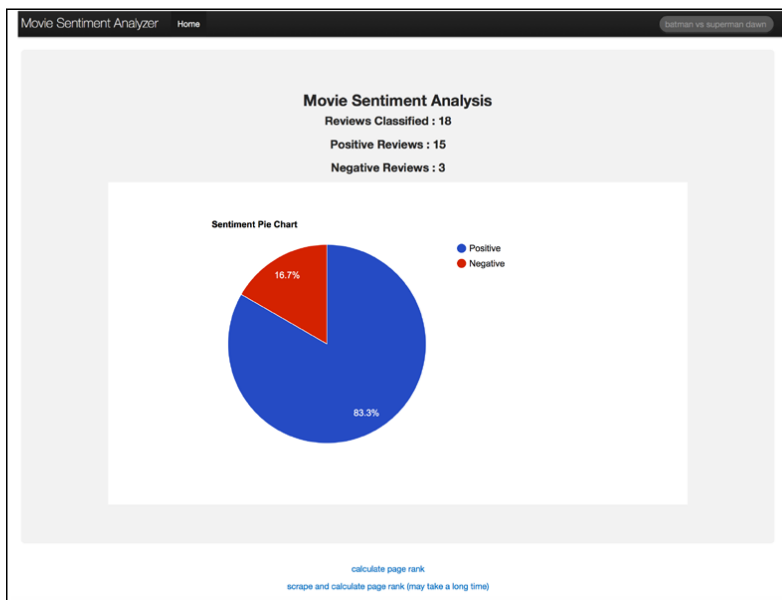


图 8.2

影评情感分析应用使用 Scrapy 库从 Bing 搜索引擎采集和抽取了 18 条影评，并分析了影评的情感倾向（15 条积极和 3 条消极倾向）。所有数据用 Django model 进行存储，便于后续用 PageRank 算法计算每个页面与查询词的相关性（这些功能的入口见图 8.2 底部的链接）。我们用 PageRank 算法得到图 8.3 结果：

图 8.3 所示为与影评查询词最相关的页面，我们将爬虫的爬取深度这一参数设置为 2（更多内容见下节）。请注意，如想获得相关性较高的网页，需要爬取成千上万个网页（图 8.3 为爬取 50 个网页得到的结果）。

我们像之前那样，启动 Django 的 Web 服务器（第 6 章和第 7 章）和主应用。首先，新建一个文件夹 movie_reviews_analyzer_app 存储代码，然后初始化 Django，命令如下：

```
mkdir movie_reviews_analyzer_app
cd  movie_reviews_analyzer_app
django-admin startproject webmining_server
python manage.py startapp startapp pages
```

①该电影的中文名为《蝙蝠侠大战超人：正义黎明》。——译者注

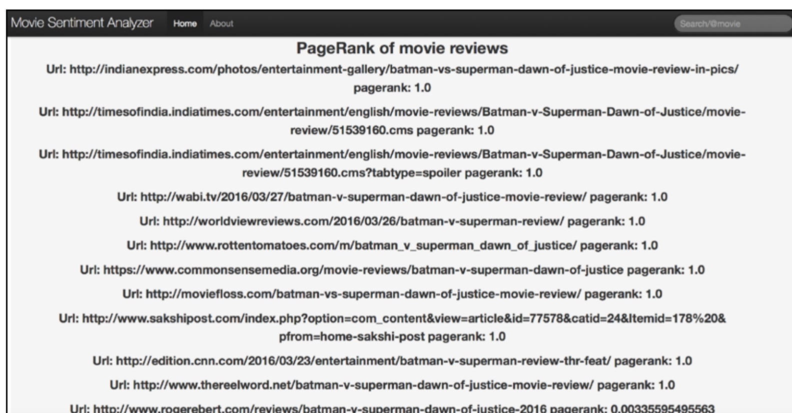


图 8.3

接着，在 settings.py 文件里做好各项配置，见 6.1.2 节和 7.1 节均讲过（当然需要注意的是，该例中项目的名称为 webmining_server 而不再是 server_movierecsys）。

情感分析应用的主视图文件在 webmining_server 文件夹，而不像之前那样位于应用（即 pages 文件夹）文件夹（见第 6 章和第 7 章），因为这些功能是服务器的通用功能，而不只是服务于特定应用（pages 应用）。

Web 服务跑起来之前，最后一项操作是，创建超级用户账号，启动 Djagno 自带的服务器：

```
python manage.py createsuperuser (admin/admin)
python manage.py runserver
```

上面解释了情感分析应用的整体结构，接下来我们从采集 URL 的搜索引擎 API 开始，更为详细地讨论该应用的各个部件。

8.2 搜索引擎的选取和应用的代码

鉴于直接从 Google、Bing、Yahoo 等搜索引擎抓取内容，违反了它们的服务条款，因此我们需要用 REST API（还可以用 Crawlera http://crawlera.com/等抓取服务）先获取一些影评页面。我们最终决定使用 Bing 提供的查询服务，它每月免费提供 5000 次查询。

使用 Bing 接口之前，注册 Microsoft Service，获取 key，搜索时需要提供该 key。这一系列操作简述如下：

注册 https://datamarket.azure.com 网站。

在 **My Account** 栏目，获取到 Primary Account Key。

注册一个新应用（在 DEVELOPERS | REGISTER；回调地址 Redirect URI 填 https://www.bing.com）。

然后，编写函数，检索与查询词相关的网页，网页数量可根据需要自己指定：

```
num_reviews = 30
def bing_api(query):
    keyBing = API_KEY       # get Bing key from: https://datamarket.
azure.com/account/keys
    credentialBing = 'Basic ' + (':%s' % keyBing).encode('base64')[:-
1] # the "-1" is to remove the trailing "\n" which encode adds
    searchString = '%27X'+query.replace(" ",'+')+'movie+review%27'
    top = 50#maximum allowed by Bing

    reviews_urls = []
    if num_reviews<top:
        offset = 0
        url = 'https://api.datamarket.azure.com/Bing/Search/Web?' + \
              'Query=%s&$top=%d&$skip=%d&$format=json' %
(searchString, num_reviews, offset)

        request = urllib2.Request(url)
        request.add_header('Authorization', credentialBing)
        requestOpener = urllib2.build_opener()
        response = requestOpener.open(request)
        results = json.load(response)
        reviews_urls = [ d['Url'] for d in results['d']['results']]
    else:
        nqueries = int(float(num_reviews)/top)+1
        for i in xrange(nqueries):
            offset = top*i
            if i==nqueries-1:
                top = num_reviews-offset
                url = 'https://api.datamarket.azure.com/Bing/Search/
Web?' + \
                      'Query=%s&$top=%d&$skip=%d&$format=json' %
(searchString, top, offset)

                request = urllib2.Request(url)
                request.add_header('Authorization', credentialBing)
                requestOpener = urllib2.build_opener()
                response = requestOpener.open(request)
            else:
```

```
                top=50
                url = 'https://api.datamarket.azure.com/Bing/Search/
Web?' + \
                      'Query=%s&$top=%d&$skip=%d&$format=json' %
(searchString, top, offset)

                request = urllib2.Request(url)
                request.add_header('Authorization', credentialBing)
                requestOpener = urllib2.build_opener()
                response = requestOpener.open(request)
            results = json.load(response)
            reviews_urls += [ d['Url'] for d in results['d']
['results']]
    return reviews_urls
```

参数 API_KEY 的值是从 Microsoft 账户得到的，query 为电影名称字符串，num_reviews = 30 为 Bing API 返回的 URL 数量。拿到了影评的 URL，我们接下来就来编写爬虫，从每个页面抽取**电影名称**和影评。

8.3 Scrapy 的配置和情感分析应用代码

Python 库 Scrapy 是用来抽取网页内容或爬取给定网页链接到的页面（更多内容见第 4 章 4.1.1 节）。如未安装 Scrapy，在终端输入以下命令进行安装：

```
sudo pip install Scrapy
```

在 bin 目录安装可执行文件：

```
sudo easy_install scrapy
```

在 movie_reviews_analyzer_app 目录下，初始化 Scrapy 项目：

```
scrapy startproject scrapy_spider
```

上述命令将在 scrapy_spider 文件夹创建以下目录文件树：

```
├── __init__.py
├── items.py
├── pipelines.py
├── settings.py
```

```
├─── spiders
|     ├─── __init__.py
```

pipelines.py 和 items.py 文件管理抓取到的数据的存储和处理方式，稍后我们会在 8.3.4 节和 8.5 节讨论这两个文件。在 settings.py 文件中，我们设置 spiders 文件夹定义的每只蜘蛛（或爬虫）的参数，蜘蛛将按照这些设置执行爬取任务。下面两节，我们将讨论爬取应用的主要参数和蜘蛛的实现。

8.3.1 Scrapy 的设置

在 settings.py 文件设置 Scrapy 项目的每只蜘蛛爬取页面时用到的所有参数。主要参数如下。

- DEPTH_LIMIT：爬取深度，从第 1 个 URL 开始，往下爬取多少层页面。默认为 0，表示没有深度限制。
- LOG_ENABLED：执行爬取时，是否将日志输出到终端，默认为 True。
- ITEM_PIPELINES = {'scrapy_spider.pipelines.ReviewPipeline': 1000,}：pipeline 函数的路径，该函数负责处理从网页抽取出来的数据。
- CONCURRENT_ITEMS = 200：pipeline 中并行处理的 item[①]的数量。
- CONCURRENT_REQUESTS = 5000：Scrapy 处理的最大并行请求数。
- CONCURRENT_REQUESTS_PER_DOMAIN = 3000：Scrapy 处理的单个域名的最大并行请求数。

深度越深，爬取页面越多，爬取时间就越长。为了加快爬取速度，上面最后三个参数可以使用较大的值。我们的应用（spiders 文件夹）使用两只蜘蛛：一只从影评 URL 抽取数据（movie_link_results.py），另一只生成所有链接到初始影评 URL 的网页链接图（recursive_link_results.py）。

8.3.2 Scraper

movie_link_results.py 的 scraper 代码如下：

```
from newspaper import Article
from urlparse import urlparse
from scrapy.selector import Selector
```

① item，用于存储抓取数据的简单容器。详见官方文档 https://doc.scrapy.org/en/latest/topics/items.html。——译者注

```
from scrapy import Spider
from scrapy.spiders import BaseSpider,CrawlSpider, Rule
from scrapy.http import Request
from scrapy_spider import settings
from scrapy_spider.items import PageItem,SearchItem

unwanted_domains = ['youtube.com','www.youtube.com']
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))

def CheckQueryinReview(keywords,title,content):
    content_list = map(lambda x:x.lower(),content.split(' '))
    title_list = map(lambda x:x.lower(),title.split(' '))
    words = content_list+title_list
    for k in keywords:
        if k in words:
            return True
    return False

class Search(Spider):
    name = 'scrapy_spider_reviews'

    def __init__ (self,url_list,search_key):#specified by -a
        self.search_key = search_key
        self.keywords = [w.lower() for w in search_key.split(" ") if w
not in stopwords]
        self.start_urls =url_list.split(',')
        super(Search, self). __init__ (url_list)

    def start_requests(self):
        for url in self.start_urls:
            yield Request(url=url, callback=self.parse_site,dont_
filter=True)

    def parse_site(self, response):
        ## Get the selector for xpath parsing or from newspaper

        def crop_emptyel(arr):
            return [u for u in arr if u!=' ']

        domain = urlparse(response.url).hostname
        a = Article(response.url)
        a.download()
```

```
        a.parse()
        title = a.title.encode('ascii','ignore').replace('\n','')
        sel = Selector(response)
        if title==None:
            title = sel.xpath('//title/text()').extract()
            if len(title)>0:
                title = title[0].encode('utf-8').strip().lower()

        content = a.text.encode('ascii','ignore').replace('\n','')
        if content == None:
            content = 'none'
            if len(crop_emptyel(sel.xpath('//div//article//p/text()').
extract()))>1:
                contents = crop_emptyel(sel.xpath('//div//article//p/
text()').extract())
                print 'divarticle'
            ….
            elif len(crop_emptyel(sel.xpath('/html/head/meta[@
name="description"]/@content').extract()))>0:
                contents = crop_emptyel(sel.xpath('/html/head/meta[@
name="description"]/@content').extract())
            content = ' '.join([c.encode('utf-8') for c in contents]).
strip().lower()

        #get search item
        search_item = SearchItem.django_model.objects.get(term=self.
search_key)
        #save item
        if not PageItem.django_model.objects.filter(url=response.url).
exists():
            if len(content) > 0:
                if CheckQueryinReview(self.keywords,title,content):
                    if domain not in unwanted_domains:
                        newpage = PageItem()
                        newpage['searchterm'] = search_item
                        newpage['title'] = title
                        newpage['content'] = content
                        newpage['url'] = response.url
                        newpage['depth'] = 0
                        newpage['review'] = True
                        #newpage.save()
                        return newpage
            else:
                return null
```

由代码可见，Search 类继承自 Scrapy 库的 Spider 类。我们编写了以下方法覆盖原有的标准方法。

- __init__：spider 的构造器，我们需要定义 start_urls 列表，以存储待抽取影评的 URL。此外，我们还在其中定义了 search_key 和 keywords 变量，用搜索引擎 API 以电影名称作为查询词查询影评数据时，将会用到这些信息。
- start_requests：调用 spider 类，将触发该函数，它指定对 start_urls 列表的每个 URL 执行什么操作；对每个 URL，调用我们自定义的 parse_site 函数（而不是默认的 parse 函数）。
- parse_site：我们自定义的函数，用来解析每个 URL 所对应页面的数据。我们用 newspaper 库（sudo pip install newspaper）抽取电影名称和文本内容，如果抽取失败，我们直接用自定义的规则，解析 HTML 文件，避免抽取到不相关的标签以至于引入噪音（每条规则用 sel .xpath 命令定义）。为了保证自定义的规则能达到较好的抽取效果，我们选择几个比较知名的域名（rottentomatoes、cnn 等）做实验，确保抽取方法能从这些网站抽取内容（上述代码没有给出所有抽取规则，更多规则还请参见 GitHub 仓库代码文件）。然后，我们用相应的 Scrapy item 和 ReviewPipeline 函数（见下节），将抽取到的数据保存到 Django model PageItem 中。
- CheckQueryinReview：我们自定义的函数，检查电影名称（来自查询词）是否存在于每个网页的内容或标题。

打开终端，进入 scrapy_spider 文件夹，输入以下命令，启动蜘蛛：

```
scrapy crawl scrapy_spider_reviews -a url_list=listname -a search_
key=keyname
```

8.3.3 Pipeline

Pipeline 定义蜘蛛抓取到新页面之后，对它进行什么操作。上面代码的 parse_site 函数返回 PageItem 对象，它触发以下 Pipeline 对象（pipelines.py）：

```
class ReviewPipeline(object):
    def process_item(self, item, spider):
        #if spider.name == 'scrapy_spider_reviews':#not working
          item.save()
          return item
```

该类保存每个 item（spider 术语，表示一个新的页面）。

8.3.4 爬虫

本章开头的综述部分提到过，我们存储了影评 URL 所有链接到的页面之后，再用 PageRank 算法计算影评的相关性。爬虫 recursive_link_results.py 执行如下操作：

```
#from scrapy.spider import Spider
from scrapy.selector import Selector
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor
from scrapy.http import Request

from scrapy_spider.items import PageItem,LinkItem,SearchItem

class Search(CrawlSpider):
    name = 'scrapy_spider_recursive'

    def __init__ (self,url_list,search_id):#specified by -a

        #REMARK is allowed_domains is not set then ALL are allowed!!!
        self.start_urls = url_list.split(',')
        self.search_id = int(search_id)

        #allow any link but the ones with different font
size(repetitions)
        self.rules = (

Rule(LinkExtractor(allow=(),deny=('fontSize=*','infoid=*','SortBy=*',
),unique=True), callback='parse_item', follow=True),
            )
        super(Search, self). __init__ (url_list)

    def parse_item(self, response):
        sel = Selector(response)

        ## Get meta info from website
        title = sel.xpath('//title/text()').extract()
        if len(title)>0:
            title = title[0].encode('utf-8')

        contents = sel.xpath('/html/head/meta[@name="description"]/@
content').extract()
        content = ' '.join([c.encode('utf-8') for c in contents]).strip()

        fromurl = response.request.headers['Referer']
```

```
        tourl = response.url
        depth = response.request.meta['depth']

        #get search item
        search_item = SearchItem.django_model.objects.get(id=self.search_id)

        #newpage
        if not PageItem.django_model.objects.filter(url=tourl).exists():
            newpage = PageItem()
            newpage['searchterm'] = search_item
            newpage['title'] = title
            newpage['content'] = content
            newpage['url'] = tourl
            newpage['depth'] = depth
            newpage.save()#cant use pipeline cause the execution can
finish here

        #get from_id,to_id
        from_page = PageItem.django_model.objects.get(url=fromurl)
        from_id = from_page.id
        to_page = PageItem.django_model.objects.get(url=tourl)
        to_id = to_page.id

        #newlink
        if not LinkItem.django_model.objects.filter(from_id=from_id).
filter(to_id=to_id).exists():
            newlink = LinkItem()
            newlink['searchterm'] = search_item
            newlink['from_id'] = from_id
            newlink['to_id'] = to_id
            newlink.save()
```

Search 类继承自 Scrapy 库的 CrawlSpider 类。我们编写了下面两个标准方法，覆盖原来的标准方法（同前面的 Spider 类）。

- __init__：Search 类的构造器。start_urls 参数指定开始爬取的 URL，在达到我们为参数 DEPTH_LIMIT 设定的值之前，爬虫将沿该 URL 往下爬取。参数 rules 设置允许/拒绝爬取的 URL 类型（该例，字体大小不同，但内容相同的页面不重复爬取）。我们还要定义处理每个抓取到的页面的函数（parse_item）。我们还定义了 search_id 变量，以便在其他数据中存储查询的 ID。
- parse_item：自定义的函数，将每个检索到的网页的重要数据存储下来。为每个页

面创建一个 Django Page model（见下节），它存储电影名称和影评内容（用 xpath HTML 解析器）。为了执行 PageRank 算法，每个网页链接到的页面和每个网页的 ID，用相应的 Scrapy item 保存为 Link model（见后面几节）。

打开终端，进入 scrapy_spider 文件夹，输入以下命令，启动爬虫：

```
scrapy crawl scrapy_spider_recursive -a url_list=listname -a search_
id=keyname
```

8.4 Django model

爬虫采集的数据需要存储到数据库。在 Django 中，数据库表称为 model，在 models.py 文件里定义（位于 pages 文件夹）。请在 models.py 文件里输入以下代码：

```
from django.db import models
from django.conf import settings
from django.utils.translation import ugettext_lazy as _

class SearchTerm(models.Model):
    term = models.CharField(_('search'), max_length=255)
    num_reviews = models.IntegerField(null=True,default=0)
    #display term on admin panel
    def __unicode__ (self):
            return self.term

class Page(models.Model):
     searchterm = models.ForeignKey(SearchTerm, related_name='pages',null
=True,blank=True)
     url = models.URLField(_('url'), default='', blank=True)
     title = models.CharField(_('name'), max_length=255)
     depth = models.IntegerField(null=True,default=-1)
     html = models.TextField(_('html'),blank=True, default='')
     review = models.BooleanField(default=False)
     old_rank = models.FloatField(null=True,default=0)
     new_rank = models.FloatField(null=True,default=1)
     content = models.TextField(_('content'),blank=True, default='')
     sentiment = models.IntegerField(null=True,default=100)

class Link(models.Model):
     searchterm = models.ForeignKey(SearchTerm, related_name='links',null
```

```
=True,blank=True)
     from_id = models.IntegerField(null=True)
     to_id = models.IntegerField(null=True)
```

用户在影评情感分析应用的首页输入的每个电影名称存储在 SearchTerm model 中，每个网页的数据存储在 Page model 中。除了记录存储内容的字段（HTML、电影名称、URL 和影评内容），还记录影评的情感倾向和链接图网络的深度（此外，我们还用一个布尔值表示网页是影评页面还是一个简单的、带外链的页面）。Link model 存储页面之间的链接关系，PageRank 算法用它计算影评页面之间的相关性。Page model 和 Link model 均通过外键指向相应的 SearchTerm。照旧执行以下命令，按照 model 生成数据表：

```
python manage.py makemigrations
python manage.py migrate
```

为了填充这些 Django model，我们需要实现 Scrapy 和 Django 之间的交互，具体方法下一节讲。

8.5　整合 Django 和 Scrapy

我们为了保持路径的简洁，便于调用，删除了 scrapy_spider 文件夹。因此，在 movie_reviews_analyzer_app 文件夹中，webmining_server 文件夹跟 scrapy_spider 文件夹位于同一层级：

```
├── db.sqlite3
├── scrapy.cfg
├── scrapy_spider
|     ├── ...
|     ├── spiders
|     |  ...
└── webmining_server
```

在 Scrapy 的 settings.py 文件里设置 Django 的路径：

```
# Setting up django's project full path.
import sys
sys.path.insert(0, BASE_DIR+'/webmining_server')
# Setting up django's settings module name.
os.environ['DJANGO_SETTINGS_MODULE'] = 'webmining_server.settings'
#import django to load models(otherwise AppRegistryNotReady: Models
```

```
aren't loaded yet):
import django
django.setup()
```

为了实现从 Scrapy 管理 Django，我们需要安装下面这个库：

```
sudo pip install scrapy-djangoitem
```

在 items.py 文件，编写 Django model 和 Scrapy 之间的对接方式：

```
from scrapy_djangoitem import DjangoItem
from pages.models import Page,Link,SearchTerm

class SearchItem(DjangoItem):
    django_model = SearchTerm
class PageItem(DjangoItem):
    django_model = Page
class LinkItem(DjangoItem):
    django_model = Link
```

上述代码，每个类继承自 DjangoItem 类，因此它们自动跟用 django_model 变量声明的、原始的 Django model 联系起来。Scrapy 项目至此告一段落，Scrapy 抓取来的数据如何处理，我们还没讲。我们接下来继续讨论处理数据用到的 Django 代码和管理应用的 Django 命令。

8.5.1 命令（情感分析模型和删除查询结果）

该应用需要管理一些操作，这些操作不允许普通用户执行，例如定义情感分析模型、删除电影查询结果，以便重新进行计算，而不是从内存检索现有数据。下面几节，我们将解释如何实现执行这些操作的命令。

8.5.2 情感分析模型加载器

该应用的最终目标是，判断影评的情感倾向（积极或消极）。为了实现这一目标，我们必须使用外部数据搭建情感分析器，然后将其存储在内存（缓存）供每次查询使用，这正是 load_sentimentclassifier.py 命令要完成的操作：

```
import nltk.classify.util, nltk.metrics
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
```

```
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from nltk.probability import FreqDist, ConditionalFreqDist
import collections
from django.core.management.base import BaseCommand, CommandError
from optparse import make_option
from django.core.cache import cache

stopwords = set(stopwords.words('english'))
method_selfeatures = 'best_words_features'

class Command(BaseCommand):
    option_list = BaseCommand.option_list + (
                make_option('-n', '--num_bestwords',
                            dest='num_bestwords', type='int',
                            action='store',
                            help=('number of words with high
information')),)

    def handle(self, *args, **options):
         num_bestwords = options['num_bestwords']
         self.bestwords = self.GetHighInformationWordsChi(num_bestwords)
         clf = self.train_clf(method_selfeatures)
         cache.set('clf',clf)
         cache.set('bestwords',self.bestwords)
```

文件开始，method_selfeatures 变量设置特征选择方法（该例用影评中的词语作为特征；更多细节见第 4 章），用选择的特征训练分类器 train_clf。参数 num_bestwords 表示最佳词语（特征）的最大数量。然后，将分类器和最佳特征（bestwords）存储到缓存，便于情感分析应用使用（用 cache 模块）。分类器和选择最佳词语（特征）的方法如下所示：

```
def train_clf(method):
    negidxs = movie_reviews.fileids('neg')
    posidxs = movie_reviews.fileids('pos')
    if method=='stopword_filtered_words_features':
        negfeatures = [(stopword_filtered_words_features(movie_
reviews.words(fileids=[file])), 'neg') for file in negidxs]
        posfeatures = [(stopword_filtered_words_features(movie_
reviews.words(fileids=[file])), 'pos') for file in posidxs]
        elif method=='best_words_features':
            negfeatures = [(best_words_features(movie_reviews.
words(fileids=[file])), 'neg') for file in negidxs]
```

```
            posfeatures = [(best_words_features(movie_reviews.
words(fileids=[file])), 'pos') for file in posidxs]
        elif method=='best_bigrams_words_features':
            negfeatures = [(best_bigrams_words_features(movie_reviews.
words(fileids=[file])), 'neg') for file in negidxs]
            posfeatures = [(best_bigrams_words_features(movie_reviews.
words(fileids=[file])), 'pos') for file in posidxs]

        trainfeatures = negfeatures + posfeatures
        clf = NaiveBayesClassifier.train(trainfeatures)
        return clf

    def stopword_filtered_words_features(self,words):
        return dict([(word, True) for word in words if word not in
stopwords])

    #eliminate Low Information Features
    def GetHighInformationWordsChi(self,num_bestwords):
        word_fd = FreqDist()
        label_word_fd = ConditionalFreqDist()

        for word in movie_reviews.words(categories=['pos']):
            word_fd[word.lower()] +=1
            label_word_fd['pos'][word.lower()] +=1

        for word in movie_reviews.words(categories=['neg']):
            word_fd[word.lower()] +=1
            label_word_fd['neg'][word.lower()] +=1

        pos_word_count = label_word_fd['pos'].N()
        neg_word_count = label_word_fd['neg'].N()
        total_word_count = pos_word_count + neg_word_count

        word_scores = {}
        for word, freq in word_fd.iteritems():
            pos_score = BigramAssocMeasures.chi_sq(label_word_fd['pos']
[word],
                (freq, pos_word_count), total_word_count)
            neg_score = BigramAssocMeasures.chi_sq(label_word_fd['neg']
[word],
                (freq, neg_word_count), total_word_count)
            word_scores[word] = pos_score + neg_score

        best = sorted(word_scores.iteritems(), key=lambda (w,s): s,
```

```
reverse=True)[:num_bestwords]
        bestwords = set([w for w, s in best])
        return bestwords

    def best_words_features(self,words):
        return dict([(word, True) for word in words if word in self.
bestwords])

    def best_bigrams_word_features(self,words,
measure=BigramAssocMeasures.chi_sq, nbigrams=200):
        bigram_finder = BigramCollocationFinder.from_words(words)
        bigrams = bigram_finder.nbest(measure, nbigrams)
        d = dict([(bigram, True) for bigram in bigrams])
        d.update(best_words_features(words))
        return d
```

上述代码实现了三种选择最佳词语的方法。

- stopword_filtered_words_features：用 NLTK（自然语言处理工具集）的停用词表，删除停用词，将剩余单词作为最相关词语。
- best_words_features：用 X^2 检验方法（NLTK 库）选择与积极影评或消极影评最相关的词语（更多细节见第 4 章）。
- best_bigrams_word_features：用 X^2 检验方法（NLTK 库）从词语组合中，选择信息量最大的前 200 个二元组（更多细节见第 4 章）。

我们用朴素贝叶斯分类器（见第 3 章）和 NLTK.corpus 的 movie_reviews 已标注的文本（积极、消极情感）作为训练语料。语料的下载方法是，打开 Python shell，输入以下代码从 corpus 下载 movie_reviews 数据：

```
nltk.download()--> corpora/movie_reviews corpus
```
[①]

8.5.3 删除已执行过的查询

由于我们可以定义不同的参数（比如特征选择方法、最佳词语的数量等），我们也许想

① 先在 Python shell 执行 nltk.download()命令（如果还没有安装 nltk，请在终端输入 pip install nltk 安装 nltk 库）。在打开的 NLTK Downloader 窗口的 Corpora 选项卡下选中 movie_reviews，单击窗口左下方的 Download 按钮下载。笔者执行完 nltk.download()命令后，显示 movie_reviews 已安装，即 Status 栏显示“installed”。——译者注

用不同的参数值，再次分析影评和存储分析结果。这时就要用到 delete_query 命令，代码如下：

```
from pages.models import Link,Page,SearchTerm
from django.core.management.base import BaseCommand, CommandError
from optparse import make_option

class Command(BaseCommand):
    option_list = BaseCommand.option_list + (
                make_option('-s', '--searchid',
                             dest='searchid', type='int',
                             action='store',
                             help=('id of the search term to delete')),)

    def handle(self, *args, **options):
         searchid = options['searchid']
         if searchid == None:
             print "please specify searchid: python manage.py
--searchid=--"
             #list
             for sobj in SearchTerm.objects.all():
                 print 'id:',sobj.id," term:",sobj.term
         else:
             print 'delete...'
             search_obj = SearchTerm.objects.get(id=searchid)
             pages = search_obj.pages.all()
             pages.delete()
             links = search_obj.links.all()
             links.delete()
             search_obj.delete()
```

如果运行上述命令时没有指定 searchid（查询词的 ID），将显示所有的查询词及其 ID。我们可以使用下面命令指定查询词并删除：

```
python manage.py delete_query --searchid=VALUE
```

我们可以使用缓存的情感分析模型向用户展示给定电影的情感倾向，详见下一节。

8.5.4 影评情感分析器——Django view 和 HTML 代码

本章讨论的大多数代码（命令、Bing 搜索引擎、Scrapy 和 Django model）是 views.py

文件 analyzer 函数的代码，它们驱动我们在 8.1 节展示的首页（在 urls.py 文件指定首页的 URL 为(r'^$','webmining_server.views.analyzer')）：

```
def analyzer(request):
    context = {}

    if request.method == 'POST':
        post_data = request.POST
        query = post_data.get('query', None)
        if query:
            return redirect('%s?%s' % (reverse('webmining_server.views.
analyzer'),
                                urllib.urlencode({'q': query})))
    elif request.method == 'GET':
        get_data = request.GET
        query = get_data.get('q')
        if not query:
            return render_to_response(
                'movie_reviews/home.html', RequestContext(request,
context))

        context['query'] = query
        stripped_query = query.strip().lower()
        urls = []

        if test_mode:
           urls = parse_bing_results()
        else:
           urls = bing_api(stripped_query)

        if len(urls)== 0:
           return render_to_response(
               'movie_reviews/noreviewsfound.html',
RequestContext(request, context))
        if not SearchTerm.objects.filter(term=stripped_query).exists():
           s = SearchTerm(term=stripped_query)
           s.save()
           try:
               #scrape
               cmd = 'cd ../scrapy_spider & scrapy crawl scrapy_spider_
reviews -a url_list=%s -a search_key=%s' %('\"'+str(','.join(urls[:num_
reviews]).encode('utf-8'))+'\"','\"'+str(stripped_query)+'\"')
               os.system(cmd)
           except:
```

```
                print 'error!'
                s.delete()
        else:
            #collect the pages already scraped
            s = SearchTerm.objects.get(term=stripped_query)

        #calc num pages
        pages = s.pages.all().filter(review=True)
        if len(pages) == 0:
            s.delete()
            return render_to_response(
                'movie_reviews/noreviewsfound.html',
RequestContext(request, context))

        s.num_reviews = len(pages)
        s.save()

        context['searchterm_id'] = int(s.id)

        #train classifier with nltk
        def train_clf(method):
            ...
        def stopword_filtered_words_features(words):
            ...
        #Eliminate Low Information Features
        def GetHighInformationWordsChi(num_bestwords):
            ...
        bestwords = cache.get('bestwords')
        if bestwords == None:
            bestwords = GetHighInformationWordsChi(num_bestwords)
        def best_words_features(words):
            ...
        def best_bigrams_words_features(words,
measure=BigramAssocMeasures.chi_sq, nbigrams=200):
            ...
        clf = cache.get('clf')
        if clf == None:
            clf = train_clf(method_selfeatures)

        cntpos = 0
        cntneg = 0
        for p in pages:
            words = p.content.split(" ")
```

```
            feats = best_words_features(words)#bigram_word_
features(words)#stopword_filtered_word_feats(words)
            #print feats
            str_sent = clf.classify(feats)
            if str_sent == 'pos':
               p.sentiment = 1
               cntpos +=1
            else:
               p.sentiment = -1
               cntneg +=1
            p.save()

        context['reviews_classified'] = len(pages)
        context['positive_count'] = cntpos
        context['negative_count'] = cntneg
        context['classified_information'] = True
    return render_to_response(
        'movie_reviews/home.html', RequestContext(request, context))
```

用户输入的电影名称先存储到 query 变量，然后程序将其发送给 bing_api 函数，去采集影评的 URL。接着调用 Scrapy，对影评 URL 执行抓取操作，找到影评的文本内容，然后用从缓存检索到的 clf 分类器模型和选定的信息量最大的词语（bestwords）执行分类（如果缓存为空，同一模型将再生成一次）。影评的情感倾向预测结果（positive_counts、negative_counts 和 reviews_classified）发送给 home.html（templates 文件夹）。首页使用下面的谷歌饼图代码生成饼图：

```
        <h2 align = Center>Movie Reviews Sentiment Analysis</h2>
        <div class="row">
        <p align = Center><strong>Reviews Classified : {{ reviews_
classified }}</strong></p>
        <p align = Center><strong>Positive Reviews : {{ positive_count
}}</strong></p>
        <p align = Center><strong> Negative Reviews : {{ negative_
count }}</strong></p>
        </div>
  <section>
      <script type="text/javascript" src="https://www.google.com/
jsapi"></script>
      <script type="text/javascript">
        google.load("visualization", "1", {packages:["corechart"]});
        google.setOnLoadCallback(drawChart);
```

```
            function drawChart() {
              var data = google.visualization.arrayToDataTable([
                ['Sentiment', 'Number'],
                ['Positive',     {{ positive_count }}],
                ['Negative',     {{ negative_count }}]
              ]);
              var options = { title: 'Sentiment Pie Chart'};
              var chart = new google.visualization.PieChart(document.
getElementById('piechart'));
              chart.draw(data, options);
           }
         </script>
           <p align ="Center" id="piechart" style="width: 900px; height:
500px;display: block; margin: 0 auto;text-align: center;" ></p>
         </div>
```

函数 drawChart 调用谷歌的可视化函数 PieChart，该函数接收数据（积极和消极影评的数量），生成饼图。HTML 代码和 Django view 交互方式的更多细节，见 6.2.2 节。首页展示影评的情感倾向统计结果（见 8.1 节），首页下方的两个链接可计算抓取到影评的 PageRank 值，该功能背后的代码下节讨论。

8.6 PageRank：Django view 和算法实现

我们在情感分析应用中实现了 PageRank 算法（4.1.3 节）为线上影评的重要性排序。pgrank.py 文件位于 webmining_server/pgrank 文件夹，它实现了 PageRank 算法，代码如下：

```
from pages.models import Page,SearchTerm

num_iterations = 100000
eps=0.0001
D = 0.85

def pgrank(searchid):
    s = SearchTerm.objects.get(id=int(searchid))
    links = s.links.all()
    from_idxs = [i.from_id for i in links ]
    # Find the idxs that receive page rank
    links_received = []
```

```
    to_idxs = []
    for l in links:
        from_id = l.from_id
        to_id = l.to_id
        if from_id not in from_idxs: continue
        if to_id not in from_idxs: continue
        links_received.append([from_id,to_id])
        if to_id not in to_idxs: to_idxs.append(to_id)

    pages = s.pages.all()
    prev_ranks = dict()
    for node in from_idxs:
        ptmp = Page.objects.get(id=node)
        prev_ranks[node] = ptmp.old_rank

    conv=1.
    cnt=0
    while conv>eps or cnt<num_iterations:
        next_ranks = dict()
        total = 0.0
        for (node,old_rank) in prev_ranks.items():
            total += old_rank
            next_ranks[node] = 0.0

        #find the outbound links and send the pagerank down to each of
them
        for (node, old_rank) in prev_ranks.items():
            give_idxs = []
            for (from_id, to_id) in links_received:
                 if from_id != node: continue
                 if to_id not in to_idxs: continue
                 give_idxs.append(to_id)
            if (len(give_idxs) < 1): continue
            amount = D*old_rank/len(give_idxs)
            for id in give_idxs:
                next_ranks[id] += amount
        tot = 0
        for (node,next_rank) in next_ranks.items():
            tot += next_rank
        const = (1-D)/ len(next_ranks)

        for node in next_ranks:
```

```
            next_ranks[node] += const

        tot = 0
        for (node,old_rank) in next_ranks.items():
            tot += next_rank

        difftot = 0
        for (node, old_rank) in prev_ranks.items():
            new_rank = next_ranks[node]
            diff = abs(old_rank-new_rank)
            difftot += diff
        conv= difftot/len(prev_ranks)
        cnt+=1
        prev_ranks = next_ranks

    for (id,new_rank) in next_ranks.items():
        ptmp = Page.objects.get(id=id)
        url = ptmp.url

    for (id,new_rank) in next_ranks.items():
        ptmp = Page.objects.get(id=id)
        ptmp.old_rank = ptmp.new_rank
        ptmp.new_rank = new_rank
        ptmp.save()
```

上述代码取到给定 SearchItem 对象对应的所有链接，实现了计算 t 时刻网页 i 的 PageRank 值的方法，$P(i)$通过递归求解下式得到：

$$P(i)_t = \frac{(1-D)}{N} + D\sum_{j=1}^{N} A_{ji} P(j)_{t-1} \quad \forall i\ 1,\cdots,N$$

其中，N 是总页面的数量，如果网页 j 指向网页 i，$A_{ji} = \frac{1}{N_j}$（N_j 为网页 j 的外链数）；否则[①]为 0。参数 D 为所谓的**阻尼系数**（上述代码将其设置为 0.85），表示按照转移矩阵 A 进行转移的概率。以迭代的方式计算上述等式，直到收敛参数（convergence parameter）eps 满足要求或达到最大迭代次数 num_iterations。首页影评情感分析结果展示完成后，单击首页下方的“scrape and calculate page rank (may take a long time)”[②]或“calculate page rank”[③]可调用该算法。这两个链接指向 views.py 文件的 pgrank_view 函数（通过 urls.py 文件的

① 原文为“N is 0”，实系错误。——译者注

② 意思是抓取和计算 PageRank 值（也许需要较长时间）。——译者注

③ 意思是计算 PageRank 值。——译者注

url(r'^pg-rank/(?P\d+)/','webmining_server.views.pgrank_view', name='pgrank_view')）：

```
def pgrank_view(request,pk):
    context = {}
    get_data = request.GET
    scrape = get_data.get('scrape','False')
    s = SearchTerm.objects.get(id=pk)

    if scrape == 'True':
        pages = s.pages.all().filter(review=True)
        urls = []
        for u in pages:
            urls.append(u.url)
        #crawl
        cmd = 'cd ../scrapy_spider & scrapy crawl scrapy_spider_recursive
-a url_list=%s -a search_id=%s' %('\"'+str(','.join(urls[:]).encode('utf-
8'))+'\"','\"'+str(pk)+'\"')
        os.system(cmd)

    links = s.links.all()
    if len(links)==0:
        context['no_links'] = True
        return render_to_response(
            'movie_reviews/pg-rank.html', RequestContext(request,
context))
    #calc pgranks
    pgrank(pk)
    #load pgranks in descending order of pagerank
    pages_ordered = s.pages.all().filter(review=True).order_by('-new_
rank')
    context['pages'] = pages_ordered

    return render_to_response(
        'movie_reviews/pg-rank.html', RequestContext(request, context))
```

上述代码调用爬虫，采集影评链接到的网页，用我们刚刚讨论的代码计算 PageRank 值。然后，将 PageRank 值展示到 pg-rank.html 页面（按 PageRank 值降序排列），见 8.1 节。因为这个函数处理数据时间较长（爬取成千上万个页面），我们编写 run_scrapelinks.py 命令，运行 Scrapy 爬虫（欢迎读者将其作为练习，自行阅读或修改这段代码）。

8.7　管理后台和 API

本章最后一部分，我们简要介绍 model 可能的后台管理操作和 API 实现方法，以便检索该情感分析应用的相关数据。我们可以在 admin.py 文件设置两个管理界面，以便查看 SearchTerm 和 Page model 的数据：

```
from django.contrib import admin
from django_markdown.admin import MarkdownField, AdminMarkdownWidget
from pages.models import SearchTerm,Page,Link

class SearchTermAdmin(admin.ModelAdmin):
    formfield_overrides = {MarkdownField: {'widget':
AdminMarkdownWidget}}
    list_display = ['id', 'term', 'num_reviews']
    ordering = ['-id']

class PageAdmin(admin.ModelAdmin):
    formfield_overrides = {MarkdownField: {'widget':
AdminMarkdownWidget}}
    list_display = ['id', 'searchterm', 'url','title','content']
    ordering = ['-id','-new_rank']

admin.site.register(SearchTerm,SearchTermAdmin)
admin.site.register(Page,PageAdmin)
admin.site.register(Link)
```

SearchTermAdmin 和 PageAdmin 类按照 ID 升序展示对象（PageAdmin 同时按照 new_rank 排序）。Page model 后台管理界面见图 8.4：

图 8.4

我们将 Link model 也添加到管理后台（admin.site.register(Link)），当然这不是必须的。除了启用管理后台，更有趣的是，我们可以设置 API 终点，实现用电影名称检索电影情感分析结果的功能。在 pages 文件夹 api.py 文件中，编写如下代码：

```
from rest_framework import views,generics
from rest_framework.permissions import AllowAny
from rest_framework.response import Response
from rest_framework.pagination import PageNumberPagination
from pages.serializers import SearchTermSerializer
from pages.models import SearchTerm,Page

class LargeResultsSetPagination(PageNumberPagination):
    page_size = 1000
    page_size_query_param = 'page_size'
    max_page_size = 10000

class SearchTermsList(generics.ListAPIView):

    serializer_class = SearchTermSerializer
    permission_classes = (AllowAny,)
    pagination_class = LargeResultsSetPagination

    def get_queryset(self):
        return SearchTerm.objects.all()

class PageCounts(views.APIView):

    permission_classes = (AllowAny,)
    def get(self,*args, **kwargs):
        searchid=self.kwargs['pk']
        reviewpages = Page.objects.filter(searchterm=searchid).
filter(review=True)
        npos = len([p for p in reviewpages if p.sentiment==1])
        nneg = len(reviewpages)-npos
        return Response({'npos':npos,'nneg':nneg})
```

PageCounts 类以要搜索的 ID（电影名称）作为参数，返回电影的情感分析结果：积极影评和消极影评的数量。要获取电影名称的 SearchTerm ID，你可以从管理后台查找或使用 API 终点 SearchTermsList；该 API 返回电影名称列表及相应的 ID。我们在 serializers.py 文件设置序列化工具 serializer：

```
from pages.models import SearchTerm
from rest_framework import serializers

class SearchTermSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = SearchTerm
        fields = ('id', 'term')
```

要调用这些 API 端点，我们可以再次使用 swagger 接口（见第 6 章）或在终端使用 curl 命令。

例如：

```
curl -X GET localhost:8000/search-list/
{"count":7,"next":null,"previous":null,"results":[{"id":24,"term":"the ma
rtian"},{"id":27,"term":"steve jobs"},{"id":29,"term":"suffragette"},{"i
d":39,"term":"southpaw"},{"id":40,"term":"vacation"},{"id":67,"term":"the
revenant"},{"id":68,"term":"batman vs superman dawn of justice"}]}
```

和

```
curl -X GET localhost:8000/pages-sentiment/68/
{"nneg":3,"npos":15}
```

8.8 小结

本章介绍了影评情感分析 Web 应用的实现方法，帮助你熟悉了我们在第 3 章、第 4 章和第 6 章学到的一些算法和库。

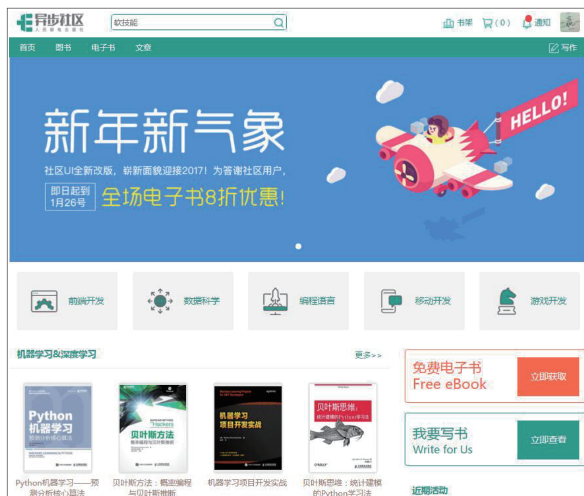
我们的旅程即将结束：通过阅读本书，运行我们提供的代码，你应该已经掌握大量当今商业环境所使用的、最为重要的机器学习算法知识。

你应该已经能够用从本书学到的知识，用 Python 语言和一些机器学习算法，开发你自己的 Web 应用。当今世界有很多数据相关的极具挑战性的问题，正等待掌握了本书所讲知识并能加以运用的人去解决。学完本书的你，当之无愧属于其中的一员。

异步社区
人 民 邮 电 出 版 社
www.epubit.com.cn

欢迎来到异步社区！

异步社区的来历



异步社区（www.epubit.com.cn）是人民邮电出版社旗下 IT 专业图书旗舰社区，于 2015 年 8 月上线运营。

异步社区依托于人民邮电出版社 20 余年的 IT 专业优质出版资源和编辑策划团队，打造传统出版与电子出版和自出版结合、纸质书与电子书结合、传统印刷与 POD 按需印刷结合的出版平台，提供最新技术资讯，为作者和读者打造交流互动的平台。

社区里都有什么？

购买图书

我们出版的图书涵盖主流 IT 技术，在编程语言、Web 技术、数据科学等领域有众多经典畅销图书。社区现已上线图书 1000 余种，电子书 400 多种，部分新书实现纸书、电子书同步出版。我们还会定期发布新书书讯。

下载资源

社区内提供随书附赠的资源，如书中的案例或程序源代码。

另外，社区还提供了大量的免费电子书，只要注册成为社区用户就可以免费下载。

与作译者互动

很多图书的作译者已经入驻社区，您可以关注他们，咨询技术问题；可以阅读不断更新的技术文章，听作译者和编辑畅聊好书背后有趣的故事；还可以参与社区的作者访谈栏目，向您关注的作者提出采访题目。

灵活优惠的购书

您可以方便地下单购买纸质图书或电子图书，纸质图书直接从人民邮电出版社书库发货，电子书提供多种阅读格式。

对于重磅新书，社区提供预售和新书首发服务，用户可以第一时间买到心仪的新书。

用户账户中的积分可以用于购书优惠。100 积分 =1 元，购买图书时，在 0 使用积分 里填入可使用的积分数值，即可扣减相应金额。

机器学习 Web 应用

Python 是一门通用型编程语言，也是一门相对容易学习的语言。因此，数据科学家在为中小规模的数据集制作原型、实现可视化和分析数据时，经常选择使用 Python。

本书填补了机器学习和 Web 开发之间的鸿沟。本书重点讲解在 Web 应用中实现预测分析功能的难点，重点介绍 Python 语言及相关框架、工具和库，展示了如何搭建机器学习系统。你将从本书学到机器学习的核心概念，学习如何将数据部署到用 Django 框架开发的 Web 应用；还将学到如何挖掘 Web、文档和服务器端数据以及如何搭建推荐引擎。

随后，你将进一步探索功能强大的 Django 框架，学习搭建一个简单、具备现代感的影评情感分析应用，它可是用机器学习算法驱动的！

本书的目标读者

本书是写给正努力成为数据科学家的读者以及新晋的数据科学家的。读者应该具备一些机器学习经验。如果你对开发智能（具备预测功能的）Web 应用感兴趣，或正在从事相关开发工作，本书非常适合你。掌握一定的 Django 知识，学习本书将会更加轻松。我们还希望读者具备一定的 Python 编程背景和扎实的统计学知识。

通过阅读本书，你将能够

- 熟悉机器学习基本概念和机器学习社区使用的一些术语。
- 用多种工具和技术从网站挖掘数据。
- 掌握 Django 框架的核心概念。
- 了解最常用的聚类和分类技术，并用 Python 实现它们。
- 掌握用 Django 搭建 Web 应用所需的所有必备知识。
- 用 Python 语言的 Django 库成功搭建和部署电影推荐系统。

异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-45852-0



分类建议：计算机/机器学习
人民邮电出版社网址：www.ptpress.com.cn